WEST Search History

DATE: Sunday, March 23, 2003

Set Name side by side	Query	Hit Count	Set Name result set
DB=USPT,PGPB,JPAB,EPAB,DWPI,TDBD; PLUR=YES; OP=ADJ			
L15	L14 and l1	11	L15
L14	multicast adj5 (radious or radii)	11	L14
L13	L12 and distributed	90	L13
L12	l3 and @AD<19990615	114	L12
L11	l2 and @AD<19990615	257	L11
L10	L9 NOT l8	1	L10
L9	l6 and @AD<19990615	3	L9
L8	l7 and @AD<19990615	2	L8
L7	L6 and unicast	46	L7
L6	L4 and (multicast or multicasting)	67	L6
L5	L4 and multicast	63	L5
L4	L3 and (locator or identifier)	267	L4
L3	L2 and (register or registering)	380	L3
L2	lookup adj5 (discovery or service or server)	716	L2
L1	lookup adj5 (discovery or service)	445	L1

END OF SEARCH HISTORY

WEST

Generate Collection

L8: Entry 2 of 2 File: DWPI Dec 21, 2000

DERWENT-ACC-NO: 2001-146817
DERWENT-WEEK: 200279

TITLE: Data processing system having lookup discovery service for accessing associated lookup services has multicast and unicast packet requests transmitted to number of lookup services

L10: Entry 1 of 1 File: USPT Nov 6, 2001

DOCUMENT-IDENTIFIER: US 6314459 B1
TITLE: Home-network autoconfiguration

DATE FILED (1):
19980813

Brief Summary Text (10):
To this end, the invention provides a method of enabling sharing a first resource or a first service, which is registered with a first information processing sub-system (e.g., a first PC), with a second information processing sub-system (e.g., a second PC). The method comprises enabling the second sub-system to be informed about access to the first sub-system and enabling registering with the second sub-system of a first interface to the first sub-system and addressable through the second sub-system. The method further comprises enabling sharing with the first sub-system, a second resource or a second service registered with the second sub-system, enabling the first sub-system to be informed about access, to the second sub-system; and enabling registering with the first sub-system of a second interface to the second sub-system and addressable through the first sub-system. Each of the sub-systems sets up as an interface proxy client for access to the other sub-system and a proxy server for handling requests from other sub-systems proxy server.

Brief Summary Text (11):
The registering of services and resources of one PC at the other one through the proxy clients thus enables automatic configuration of a network in order to share resources. The registering hides the idea whether a resource or a service is local or is residing at another apparatus. In other words, the invention uses the registering as a tool for autoconfiguration of a network.

Brief Summary Text (12):
Jini focuses on the process of adding a device to the network and broadcasting information about the device to other machines. In this way, Jini provides a "Lookup" service that allows applications on other machines to use the newly added device. The approach of Jini assumes the network and operating system have already been configured so that each computer already knows about other computers. Jini's functionality occurs at a layer above the network. It does not, for example, solve the problems of automatic configuration of the network upon connection, disconnection, or reconnection. It assumes that the network is up or down, independent of Jini. Jini leverages the services provided by the network to implement its services. In other words, the invention uses the registering as a tool for autoconfiguration

Detailed Description Text (4):
PC 102 has a registry 124 for registering interfaces to resources and services 108-114 local to PC 102. Applications running on PC 102 can get access to these interfaces. The interfaces handle the messages or requests to local resources or services 108-114. Similarly, PC 104 has a registry 126 for registering interfaces to resources and services 116-122 local to PC 104.

Detailed Description Text (8):
FIG. 4 shows a further step in the autoconfiguration process. Port listener 130 starts up a proxy server 144 to handle requests from remote client 140. Proxy server 144 sends information about resource 108-114, e.g., as contained in registry 124, to proxy client 140. Proxy client 140 registers this information with registry 126. Similarly, port listener 136 starts up a proxy server 146 to handle requests from remote client 142. Proxy server 146 sends information about resources 116-122 to proxy client 142, which thereupon registers this with registry 124.

Detailed Description Text (9):

FIG. 5 shows the step wherein proxy client 142 registers with registry 124 as a local service for each resource or service available in registry 126, and wherein proxy client 140 registers with registry 126. for each resource or service available in registry 124. The result is that PC 102 has now copied its registry 124 specifying the addresses of its local resources and services to PC 104, where the copy is added to registry 126. Similarly, PC 104 has copied its registry 126 to PC 102 where it gets added to registry 124 Both PC's 102 and 104 have now been registered with each other. When a third PC 148 is connected to bus 106, a process similar as the one discussed above occurs automatically. Registries 124 and 126 thus hide the idea of whether a resource or a service is local or is residing at another apparatus. Each of the addresses in registry 124 is unique throughout registry 124. Similarly, each of the addresses in registry 126 is unique throughout registry 126. A user working at PC 102 and requesting a local resource or service, i.e., one of resources or services 108-114 has the request passed on directly to the resource or service requested indicated by the corresponding address in registry 124. When the user requests a remote service or remote resource, i.e., one of resources or services 116-122 that is local to remote PC 104, the request is forwarded to proxy client 142 and processed by proxy server 146 as discussed below with reference to FIG.7.

Detailed Description Text (10):
FIG. 6 gives diagrams 600 and 602 as an example of the registering of proxy client 142 with registry 124. Diagram 600 represents initial registry 126 with a list of local resources and services available at PC 104 and their respective local addresses. Diagram 602 represents registry 124 after client 142 has registered with it. Registry 124 initially comprises the list of resources and services 108-114 with local addresses #1 to #K. After client 142 has registered, registry 124 has an entry for PC 104 as proxy device at address #Q. Remote resources and services 116-122 now have addresses dependent on address #Q.

Detailed Description Text (15):
An implementation of this autoconfiguration system can be demonstrated using a java-based system. In this scenario, two stand-alone PC's have java, a network card, and a TCP/IP stack installed on their machines. For purposes of using TCP/IP, a respective random IP number is used for each respective one of the machines. The idea here is that the IP address and standard TCP/IP settings have not been pre-configured by the user. In addition to this IP address, a unique id (UID) is generated--this could be a random number of sufficient complexity so that a collision is statistically improbable (a working exams of this is Microsoft's Global Unique IDentifier [GUID]). The IP address is used to allow TCP/IP to identify each PC and the UID is used to ensure that the randomly generated IPs are not the same. This is established since the same IP address must necessarily be connected to the same UID.

Detailed Description Text (16):
In the java system, a runtime executable is started that spawns off three objects: a Broadcaster, a Broadcast Listener, and a Port Listener--each as a java thread. In this case, the Broadcast Listener could use a java multicast socket to subscribe to a multicast. Likewise, the Broadcaster could also use a java multicast socket to send information to a multicast group. The Port Listener could be implemented as a java ServerSocket.

CLAIMS:

1. An information processing system (100) with a first information processing sub-system (102) coupled to a second information processing sub-system (104), wherein:

the first sub-system has a first registry (124) for registering at least a first resource (108-114) local to the first sub-system;

the second sub-system has a second registry (126) for registering at least a second resource (116-122) local to the second sub-system;

the first sub-system has a first proxy client (142) registered with the first registry; and

the second sub-system has a second proxy server (146) for communicating with the first proxy client and for access of the second resource.

4. An information processing sub-system (102) comprising:

a registry (124) for registering a resource (108-114) local to the sub-system;

a broadcasting module (128) for broadcasting a message;

a broadcast listener (132) for receiving a response from another information processing sub-system in response to the other sub-system having received the message, and for creating a proxy client (142) thereupon;

wherein:

the proxy client is being registered with the registry as a representative of the other sub-system for the sub-system to access another resource local to the other sub-system.

7. A method for enabling sharing with a first information processing sub-system (102) a second resource (116-122) registered with a second information processing sub-system (104), the method comprising:

enabling the first sub-system to be informed about access to the second sub-system; and

enabling registering with a registry of resources that are local to the first sub-system a second interface to the second resource for access to the second resource from the first sub-system.

9. The method of claim 7, further comprising:

enabling sharing with the second sub-system a first resource (108-114) registered with the first sub-system;

enabling the second sub-system to be informed about access to the first sub-system; and

enabling registering with the second sub-system of a first interface to the first resource for access to t first resource from the second sub-system.

13. A method for enabling a first information processing sub-system (102) to address a resource (116-122) registered with a second information processing sub-system (104), the method comprising:

enabling creating a proxy client (142) at the first sub-system for communication with the second sub-system, the proxy client being representative of the resource;

enabling registering the proxy client as a local resource with a registry (124) for further resources local to the first sub-system;

enabling creating a proxy server (146) at the second information processing sub-system for handling a request from the proxy client.

14. The method of claim 13, comprising enabling the second information processing sub-system to address a further resource (108-114) registered with the first information processing sub-system, the method further comprising:

enabling creating a further proxy client (140) at the second sub-system for communication with the first sub-system, the further proxy client being representative of the further resource;

enabling registering the further proxy client as a further local resource with a further registry (126) of the second sub-system;

enabling creating a further proxy server (144) at the first information processing sub-system for handling a further request from the further proxy client.

17. A method of enabling a user to configure a home network with a first information processing sub-system and a second information processing sub-system, wherein:

a first resource local to the first sub-system is registered with a first registry of the first sub-system;

a second resource local to the second sub-system is registered with a second registry of the second sub-system; and

the method comprises:

enabling creating a proxy client at the first sub-system for communication with the second sub-system, the proxy client being representative of the second resource;

enabling registering the proxy client as a local resource with the first registry; and

enabling creating a proxy server at the second information processing sub-system for handling a request from the proxy client.

18. A software application for implementing a method of enabling a user to configure a home network with a first information processing sub-system and a second information processing sub-system, wherein:

a first resource local to the first sub-system is registered with a first registry of the first sub-system;

a second resource local to the second sub-system is registered with a second registry of the second sub-system; and

the method comprises:

enabling to create a proxy client at the first sub-system for communication with the second sub-system, the proxy client being representative of the second resource;

enabling to register the proxy client as a local resource with the first registry; and

enabling to create a proxy server at the second information processing sub-system for handling a request from the proxy client.

WEST

Generate Collection

L13: Entry 1 of 90 File: PGPB Feb 7, 2002

DOCUMENT-IDENTIFIER: US 20020016790 A1
TITLE: APPARATUS AND METHOD FOR DYNAMICALLY VERIFYING INFORMATION IN A DISTRIBUTED SYSTEM

Application Filing Date (1):
19980320

Abstract Paragraph (1):
Use of a policy object for verification in a distributed system. A machine downloads a policy object containing a reference to code governing verification of data. The machine uses the reference to obtain the code and locally verify data or other information. As particular rules for the data change, the policy object may be updated to provide a reference to the code for the new rules when it is downloaded.

Summary of Invention Paragraph (2):
[0003] The present invention relates to a system and method for transmitting objects between machines in a distributed system and more particularly to dynamically verifying information in a distributed system.

Summary of Invention Paragraph (4):
[0004] Distributed programs which concentrate on point-to-point data transmission can often be adequately and efficiently handled using special-purpose protocols for remote terminal access and file transfer. Such protocols are tailored specifically to the one program and do not provide a foundation on which to build a variety of distributed programs (e.g., distributed operating systems, electronic mail systems, computer conferencing systems, etc.).

Summary of Invention Paragraph (5):
[0005] While conventional transport services can be used as the basis for building distributed programs, these services exhibit many organizational problems, such as the use of different data types in different machines, lack of facilities for synchronization, and no provision for a simple programming paradigm.

Summary of Invention Paragraph (6):
[0006] Distributed systems usually contain a number of different types of machines interconnected by communications networks. Each machine has its own internal data types, its own address alignment rules, and its own operating system. This heterogeneity causes problems when building distributed systems. As a result, program developers must include in programs developed for such heterogeneous distributed systems the capability of dealing with ensuring that information is handled and interpreted consistently on different machines.

Summary of Invention Paragraph (16):
[0016] Because the JVM may be implemented on any type of platform, implementing distributed programs using the JVM significantly reduces the difficulties associated with developing programs for heterogenous distributed systems. Moreover, the JVM uses a Java remote method invocation system (RMI) that enables communication among programs of the system. RMI is explained in, for example, the following document, which is incorporated herein by reference: Remote Method Invocation Specification, Sun Microsystems, Inc. (1997), which is available via universal resource locator (URL) http://www.javasoft.com/products/jdk/1.1- /docs/guide/rmi/spec/rmiTOC.doc.html.

Summary of Invention Paragraph (17):
[0017] FIG. 2 is a diagram illustrating the flow of objects in an object-oriented distributed system 200 including machines 201 and 202 for transmitting and receiving method invocations using the JVM. In system 200, machine 201 uses RMI 205 for responding to a call for object 203 by converting the object into a byte stream 207

including an identification of the type of object transmitted and data constituting the object. While machine 201 is responding to the call for object 203, a process running on the same or another machine in system 200 may continue operation without waiting for a response to its request.

Summary of Invention Paragraph (19):
[0019] The communication among the machines may include verification of data or other information. Such verification typically requires multiple calls for verification of particular data or other information, which may result in a large volume of calls and potentially increased expense for the verification. Accordingly, a need exists for efficient verification of data or other information in a distributed system.

Brief Description of Drawings Paragraph (4):
[0028] FIG. 2 is a diagram illustrating the transmission of objects in an object-oriented distributed system;

Brief Description of Drawings Paragraph (5):
[0029] FIG. 3 is a diagram of an exemplary distributed processing system that can be used in an implementation consistent with the present invention;

Brief Description of Drawings Paragraph (6):
[0030] FIG. 4 is a diagram of an exemplary distributed system infrastructure;

Brief Description of Drawings Paragraph (7):
[0031] FIG. 5 is a diagram of a computer in a distributed system infrastructure shown in FIG. 4;

Brief Description of Drawings Paragraph (8):
[0032] FIG. 6 is a diagram of an exemplary distributed network for use in transmission of a policy object; and

Detail Description Paragraph (3):
[0034] Machines consistent with the present invention may use a policy object, also referred to as a verification object, in a distributed system, the policy object performing processing when verification is needed. A machine downloads a policy object containing a reference to code governing verification of data or other information. The machine uses the reference to obtain the code and locally verify, for example, data constraints among items, data items, or objects. A verification object may also be used to verify other types of information. As particular rules for the data or information change, the policy object may be updated to provide a reference to the code for the new rules when it is downloaded.

Detail Description Paragraph (4):
[0035] Systems consistent with the present invention may efficiently transfer policy objects using a variant of an RPC or RMI, passing arguments and return values from one process to another process each of which may be on different machines. The term "machines" is used in this context to refer to a physical machine or a virtual machine. Multiple virtual machines may exist on the same physical machine. Examples of RPC systems include distributed computed environment (DCE) RPC and Microsoft distributed common object model (DCOM) RPC.

Detail Description Paragraph (5):
Distributed Processing Systems

Detail Description Paragraph (6):
[0036] FIG. 3 illustrates an exemplary distributed processing system 300 which can be used in an implementation consistent with the present invention. In FIG. 3, distributed processing system 300 contains three independent and heterogeneous platforms 301, 302, and 303 connected in a network configuration represented by network cloud 319. The composition and protocol of the network configuration represented by cloud 319 is not important as long as it allows for communication of the information between platforms 301, 302 and 303. In addition, the use of just three platforms is merely for illustration and does not limit an implementation consistent with the present invention to the use of a particular number of platforms. Further, the specific network architecture is not crucial to embodiments consistent with this invention. For example, another network architecture that could be used in an implementation consistent with this invention would employ one platform as a network controller to which all the other platforms would be connected.

Detail Description Paragraph (7):
[0037] In the implementation of distributed processing system 300, platforms 301, 302 and 303 each include a processor 316, 317, and 318 respectively, and a memory, 304, 305, and 306, respectively. Included within each memory 304, 305, and 306, are applications 307, 308, and 309, respectively, operating systems 310, 311, and 312, respectively, and RMI components 313, 314, and 315, respectively.

Detail Description Paragraph (12):
Distributed System Infrastructure

Detail Description Paragraph (13):
[0042] Systems and methods consistent with the present invention may also operate within a particular distributed system 400, which will be described with reference to FIGS. 4 and 5. This distributed system 400 is comprised of various components, including hardware and software, to (1) allow users of the system to share services and resources over a network of many devices; (2) provide programmers with tools and programming patterns that allow development of robust, secured distributed systems; and (3) simplify the task of administering the distributed system. To accomplish these goals, distributed system 400 utilizes the Java programming environment to allow both code and data to be moved from device to device in a seamless manner. Accordingly, distributed system 400 is layered on top of the Java programming environment and exploits the characteristics of this environment, including the security offered by it and the strong typing provided by it.

Detail Description Paragraph (14):
[0043] In distributed system 400 of FIGS. 4 and 5, different computers and devices are federated into what appears to the user to be a single system. By appearing as a single system, distributed system 400 provides the simplicity of access and the power of sharing that can be provided by a single system without giving up the flexibility and personalized response of a personal computer or workstation. Distributed system 400 may contain thousands of devices operated by users who are geographically disperse, but who agree on basic notions of trust, administration, and policy.

Detail Description Paragraph (15):
[0044] Within an exemplary distributed system are various logical groupings of services provided by one or more devices, and each such logical grouping is known as a Djinn. A "service" refers to a resource, data, or functionality that can be accessed by a user, program, device, or another service and that can be computational, storage related, communication related, or related to providing access to another user. Examples of services provided as part of a Djinn include devices, such as printers, displays, and disks; software, such as programs or utilities; information, such as databases and files; and users of the system.

Detail Description Paragraph (17):
[0046] Distributed system 400 is comprised of computer 402, a computer 404, and a device 406 interconnected by a network 408. Device 406 may be any of a number of devices, such as a printer, fax machine, storage device, computer, or other devices. Network 408 may be a local area network, wide area network, or the Internet. Although only two computers and one device are depicted as comprising distributed system 400, one skilled in the art will appreciate that distributed system 400 may include additional computers or devices.

Detail Description Paragraph (18):
[0047] FIG. 5 depicts computer 402 in greater detail to show a number of the software components of distributed system 400. One skilled in the art will appreciate that computer 404 or device 406 may be similarly configured. Computer 402 includes a memory 502, a secondary storage device 504, a central processing unit (CPU) 506, an input device 508, and a video display 510. Memory 502 includes a lookup service 512, a discovery server 514, and a Java runtime system 516. The Java runtime system 516 includes the Java RMI system 518 and a JVM 520. Secondary storage device 504 includes a Java space 522.

Detail Description Paragraph (19):
[0048] As mentioned above, distributed system 400 is based on the Java programming environment and thus makes use of the Java runtime system 516. The Java runtime system 516 includes the Java API libraries, allowing programs running on top of the Java runtime system to access, in a platform-independent manner, various system functions, including windowing capabilities and networking capabilities of the host operating system. Since the Java API libraries provides a single common API across all operating

systems to which the Java runtime system is ported, the programs running on top of a Java runtime system run in a platform-independent manner, regardless of the operating system or hardware configuration of the host platform. The Java runtime system 516 is provided as part of the Java software development kit available from Sun Microsystems, Inc. of Mountain View, Calif.

Detail Description Paragraph (21):
[0050] Lookup service 512 defines the services that are available for a particular Djinn. That is, there may be more than one Djinn and, consequently, more than one lookup service within distributed system 400. Lookup service 512 contains one object for each service within the Djinn, and each object contains various methods that facilitate access to the corresponding service. Lookup service 512 is described in U.S. patent application entitled "Method and System 415 for Facilitating Access to a Lookup Service," which was previously incorporated herein by reference.

Detail Description Paragraph (22):
[0051] Discovery server 514 detects when a new device is added to distributed system 400, during a process known as boot and join (or discovery), and when such a new device is detected, the discovery server passes a reference to lookup service 512 to the new device so that the new device may register its services with the lookup service and become a member of the Djinn. After registration, the new device becomes a member of the Djinn, and as a result, it may access all the services contained in lookup service 512. The process of boot and join is described in U.S. patent application entitled "Apparatus and Method for providing Downloadable Code for Use in Communicating with a Device in a Distributed System," which was previously incorporated herein by reference.

Detail Description Paragraph (23):
[0052] A Java space 522 is an object repository used by programs within distributed system 400 to store objects. Programs use a Java space 522 to store objects persistently as well as to make them accessible to other devices within distributed system 400. Java spaces are described in U.S. patent application Ser. No. 08/971,529, entitled "Database System Employing Polymorphic Entry and Entry Matching," assigned to a common assignee, and filed on Nov. 17, 1997, which is incorporated herein by reference. One skilled in the art will appreciate that an exemplary distributed system 400 may contain many lookup services, discovery servers, and Java spaces.

Detail Description Paragraph (24):
Data Flow in a Distributed Processing System

Detail Description Paragraph (25):
[0053] FIG. 6 is a diagram of an object-oriented distributed network 600 for use in transmission of a policy object for use in verification. Network 600 includes client machine 601 and server machine 604, which may be implemented with computers or virtual machines executing on one or more computers, or the machines described with reference to FIGS. 3, 4, and 5. Client machine 601 includes RMI 602 and associated object 603. Server machine 604 includes RMI 605 and associated policy object 606.

Detail Description Paragraph (30):
[0057] FIG. 7 is a flow chart of a process 700 for verification using a policy object, also referred to as a verification object. A machine first determines if verification is requested (step 701). If so, it requests a policy object from a server (step 702) and receives the policy object including a reference to code for use in verification of data or other information (step 703). Using the reference, it downloads code for the verification (step 704). The downloading of code may occur using the methods described in U.S. patent application Ser. No. 08/950,756, filed on Oct. 15, 1997, and entitled "Deferred Reconstruction of Objects and Remote Loading in a Distributed System," which is incorporated herein by reference.

Detail Description Paragraph (32):
[0059] Machines implementing the steps shown in FIG. 7 may include computer processors for performing the functions, as shown in FIGS. 3, 4, 5, and 6. They may include modules or programs configured to cause the processors to perform the above functions. They may also include computer program products stored in a memory. The computer program products may include a computer-readable medium or media having computer-readable code embodied therein for causing the machines to perform functions described in this specification. The media may include a computer data signal embodied in a carrier wave and representing sequences of instructions which, when executed by a processor, cause the processor to securely address a peripheral device at an absolute

address by performing the method described in this specification. The media may also include data structures for use in performing the method described in this specification. In addition, the processing shown in FIG. 7 may occur through the use of smart proxies, which are described in U.S. patent application filed on even date herewith, assigned to a common assignee, and entitled "Downloadable Smart Proxies for Performing Processing Associated with a Remote Procedure Call in a Distributed System," which is incorporated herein by reference.

CLAIMS:

1. A method for transmitting objects in a distributed system comprised of multiple machines, comprising: transmitting a request for a verification object; receiving a response to the request including an indication of a first code corresponding to the verification object and an indication of a second code for processing associated with verification; constructing the verification object using the indicated first code; and verifying information using the indicated second code.

4. A method for transmitting objects in a distributed system comprised of multiple machines, comprising: transmitting a request for a verification object; receiving a response to the request including an indication of a code corresponding to the verification object; constructing the verification object using the indicated code; and verifying information based on the constructed object.

5. A method for transmitting objects in a distributed system comprised of multiple machines, comprising: receiving at a machine a request for an object for use in verification; and transmitting a response to the request including an indication of a first code for constructing the verification object and including an indication of a second code for processing associated with the verification.

6. An apparatus for transmitting objects in a distributed system comprised of multiple machines, comprising: a module configured to transmit a request for a verification object; a module configured to receive a response to the request including an indication of a first code corresponding to the verification object and an indication of a second code for processing associated with verification; a module configured to construct the verification object using the indicated first code; and a module configured to verifying information using the indicated second code.

9. An apparatus for transmitting objects in a distributed system comprised of multiple machines, comprising: a module configured to transmit a request for a verification object; a module configured to receive a response to the request including an indication of a code corresponding to the verification object; a module configured to construct the verification object using the indicated code; and a module configured to verify information based on the constructed object.

10. An apparatus for processing objects in a distributed system comprised of multiple machines, comprising: a module configured to receive at a machine a request for an object for use in verification; and a module configured to transmit a response to the request including an indication of a first code for constructing the verification object and including an indication of a second code for processing associated with the verification.

11. A system for transmitting objects in a distributed system comprised of multiple machines, comprising: a first machine; a second machine; a network connecting the first machine with the second machine; and an apparatus for receiving objects, the apparatus including: a module configured to transmit a request for a verification object; a module configured to receive a response to the request including an indication of a first code corresponding to the verification object and an indication of a second code for processing associated with verification; a module configured to construct the verification object using the indicated first code; and a module configured to verify information using the indicated second code.

14. A system for transmitting objects in a distributed system comprised of multiple machines, comprising: a first machine; a second machine; a network connecting the first machine with the second machine; and an apparatus for receiving objects, the apparatus including: a module configured to transmit a request for a verification object; a module configured to receive a response to the request including an indication of a code corresponding to the verification object; a module configured to construct the verification object using the indicated code; and a module configured to verify information based on the constructed object.

15. A system for transmitting objects in a distributed system comprised of multiple machines, comprising: a first machine; a second machine; a network connecting the first machine with the second machine; and an apparatus for transmitting objects, the apparatus including: a module configured to receive at a machine a request for an object for use in verification; and a module configured to transmit a response to the request including an indication of a first code for constructing the verification object and including an indication of a second code for processing associated with the verification.

21. An article of manufacture specifying a representation of an object stored in a computer-readable storage medium and capable of electronic transmission between machines in a distributed system, the article of manufacture comprising: a verification object to be transmitted from a first machine to a second machine in response to a request, the verification object including an indication of code for performing processing for verification of information.

22. An apparatus for transmitting objects in a distributed system comprised of multiple machines, comprising: means for transmitting a request for a verification object; means for receiving a response to the request including an indication of a first code corresponding to the verification object and an indication of a second code for processing associated with verification; means for constructing the verification object using the indicated first code; and verifying information using the indicated second code.

L13: Entry 4 of 90 File: PGPB Nov 15, 2001

DOCUMENT-IDENTIFIER: US 20010042091 A1
TITLE: POLYMORPHIC TOKEN BASED CONTROL

Application Filing Date (1):
19980320

Summary of Invention Paragraph (3):
[0002] Provisional U.S. Patent Application No._____ , entitled "Distributed Computing System," filed on Feb. 26, 1998.

Summary of Invention Paragraph (5):
[0004] U.S. patent application Ser. No._____ , entitled "Method, Apparatus, and Product for Leasing of Delegation Certificates in a Distributed System," bearing attorney docket no. 06502.0011-02000, and filed on the same date herewith.

Summary of Invention Paragraph (6):
[0005] U.S. patent application Ser. No._____ , entitled "Method, Apparatus and Product for Leasing of Group Membership in a Distributed System," bearing attorney docket no. 06502.0011-03000, and filed on the same date herewith.

Summary of Invention Paragraph (9):
[0008] U.S. patent application Ser. No._____, entitled "Deferred Reconstruction of Objects and Remote Loading for Event Notification in a Distributed System," bearing attorney docket no. 06502.0062-01000, and filed on the same date herewith.

Summary of Invention Paragraph (12):
[0011] U.S. patent application Ser. No._____, entitled "Method and Apparatus for Determining Status of Remote Objects in a Distributed System," bearing attorney docket no. 06502.0104-00000, and filed on the same date herewith.

Summary of Invention Paragraph (13):
[0012] U.S. patent application Ser. No. . entitled "Downloadable Smart Proxies for Performing Processing Associated with a Remote Procedure Call in a Distributed System," bearing attorney docket no. 06502.0105-00000, and filed on the same date herewith.

Summary of Invention Paragraph (18):
[0017] U.S. patent application Ser. No._____, entitled "Dynamic Lookup Service in a Distributed System," bearing attorney docket no. 06502.0110-00000, and filed on the same date herewith.

Summary of Invention Paragraph (19):
[0018] U.S. patent application Ser. No._____, entitled "Apparatus and Method for Providing Downloadable Code for Use in Communicating with a Device in a Distributed System," bearing attorney docket no. 06502.0112-00000, and filed on the same date herewith.

Summary of Invention Paragraph (20):
[0019] U.S. patent application Ser. No._____, entitled "Method and System for Facilitating Access to a Lookup Service," bearing attorney docket no. 06502.0113-00000, and filed on the same date herewith.

Summary of Invention Paragraph (21):
[0020] U.S. patent application Ser. No._____, entitled "Apparatus and Method for Dynamically Verifying Information in a Distributed System," bearing attorney docket no. 06502.0114-00000, and filed on the same date herewith.

Summary of Invention Paragraph (22):

[0021] U.S. patent application Ser. No. 09/030,840, entitled "Method and Apparatus for Dynamic Distributed Computing Over a Network," and filed on Feb. 26, 1998.

Detail Description Paragraph (5):
[0046] FIG. 1 is a high level diagram of a token ring network 100 made up of four distributed computers 102, 104, 106, and 108 passing a token object in the counter clockwise direction through network media 120. The token object is preferably passed between computers 102-108 using some form of remote object passing mechanism, such as the Java remote invocation system (Java RMI). Additionally, one of computers 102-108 may act as a gateway to a larger token ring network or to a non token ring network. As shown in FIG. 1, computer 106 acts as a gateway to the Internet network 110.

Detail Description Paragraph (6):
[0047] In exemplary distributed system 100, different computers and devices are federated into what appears to the user to be a single system. By appearing as a single system, the distributed system 100 provides the simplicity of access and the power of sharing that can be provided by a single system without giving up the flexibility and personalized response of a personal computer or workstation. Distributed system 100 may contain thousands of devices operated by users who are geographically disperse, but who agree on basic notions of trust, administration, and policy.

Detail Description Paragraph (7):
[0048] Within the distributed 100 system are various logical groupings of services provided by one or more devices, and each such logical grouping is known as a Djinn. A "service" refers to a resource, data, or functionality that can be accessed by a user, program, device, or another service and that can be computational, storage related, communication related, or related to providing access to another user. Examples of services provided as part of a Djinn include devices, such as printers, displays, and disks; software, such as applications or utilities; information, such as databases and files; and users of the system.

Detail Description Paragraph (10):
[0051] FIG. 2 depicts computer 102 in greater detail showing a number of the software components of the distributed system 100. Computer 102 includes a memory 202, a secondary storage device 204, a central processing unit (CPU) 206, an input device 208, and a video display 210. The memory 202 includes a lookup service 212, a discovery server 214, and a Java.TM. runtime system 216. The Java runtime system 216 includes the Java.TM. remote method invocation system (RMI) 218 and a Java.TM. virtual machine 220. The secondary storage device 204 includes a JavaSpace.TM. 222.

Detail Description Paragraph (11):
[0052] The exemplary distributed system 100 is based on the Java programming environment and thus makes use of the Java runtime system 216. The Java runtime system 216 includes the Java API, allowing programs running on top of the Java runtime system to access, in a platform-independent manner, various system functions, including windowing capabilities and networking capabilities of the host operating system. Since the Java API provides a single common API across all operating systems to which the Java runtime system is ported, the programs running on top of a Java runtime system run in a platform-independent manner, regardless of the operating system or hardware configuration of the host platform. The Java runtime system 216 is provided as part of the Java software development kit available from Sun Microsystems of Mountain View, Calif.

Detail Description Paragraph (13):
[0054] The lookup service 212 defines the services that are available for a particular Djinn. That is, there may be more than one Djinn and, consequently, more than one lookup service within the exemplary distributed system 100. The lookup service 212 contains one object for each service within the Djinn, and each object contains various methods that facilitate access to the corresponding service. The lookup service 212 and its access are described in greater detail in co-pending U.S. patent application Ser. No._____, entitled "Method and System for Facilitating Access to a Lookup Service," which has been previously incorporated by reference.

Detail Description Paragraph (14):
[0055] The discovery server 214 detects when a new device is added to the exemplary distributed system 100, during a process known as boot and join or discovery, and when such a new device is detected the discovery server passes a reference to the lookup service 212 to the new device so that the new device may register its services with the lookup service and become a member of the Djinn. After registration, the new device

becomes a member of the Djinn, and as a result, it may access all the services contained in the lookup service 212. The process of boot and join is described in greater detail in co-pending U.S. patent application Ser. No._____, entitled "Apparatus and Method for providing Downloadable Code for Use in Communicating with a Device in a Distributed System," which has previously been incorporated by reference.

Detail Description Paragraph (15):
[0056] The JavaSpace 222 is an object repository used by programs within the exemplary distributed system 100 to store objects. Programs use the JavaSpace 222 to store objects persistently as well as to make them accessible to other devices within the exemplary distributed system. Java spaces are described in greater detail in co-pending U.S. patent application Ser. No. 08/971,529, entitled "Database System Employing Polymorphic Entry and Entry Matching," assigned to a common assignee, filed on Nov. 17, 1997, which is incorporated herein by reference. One skilled in the art will appreciate that the exemplary distributed system 100 may contain many lookup services, discovery servers, and JavaSpaces.

Detail Description Paragraph (16):
[0057] Although systems and methods consistent with the present invention are described as operating in the exemplary distributed system and the Java programming environment, one skilled in the art will appreciate that the present invention can be practiced in other systems and other programming environments. Additionally, although aspects of the present invention are described as being stored in memory, one skilled in the art will appreciate that these aspects can also be stored on or read from other types of computer-readable media, such as secondary storage devices, like hard disks, floppy disks, or CD-ROM; a carrier wave from the Internet; or other forms of RAM or ROM. Sun, Sun Microsystems, the SunLogo, Java, and Java-based trademarks are trademarks or registered trademarks of Sun Microsystems Inc. in the United States and other countries.

Detail Description Paragraph (18):
[0058] A token ring network consistent with the present invention passes a polymorphic token object around the network in place of the static token frame used in conventional token ring networks. The passing of the token object is preferably implemented using a distributed object-oriented programming environment, such as Java RMI (described above). Java RMI is especially suitable to the present invention, as it provides for the automatic management of distributed objects and the ability to easily pass objects from machine to machine on a network.

WEST

Generate Collection

L13: Entry 8 of 90 File: USPT Nov 26, 2002

DOCUMENT-IDENTIFIER: US 6487607 B1
TITLE: Methods and apparatus for remote method invocation

DATE FILED (1):
19980320

Parent Case Text (3):
Provisional U.S. Patent Application No. 60/076,048, entitled "Distributed Computing System," filed on Feb. 26, 1998.

Parent Case Text (5):
U.S. patent application Ser. No. 09/044,838, entitled "Method, Apparatus, and Product for Leasing of Delegation Certificates in a Distributed System," filed on Mar. 20, 1998, now U.S. Pat. No. 6,247,026.

Parent Case Text (6):
U.S. patent application Ser. No. 09/044,834, entitled "Method, Apparatus and Product for Leasing of Group Membership in a Distributed System," and filed on Mar. 20, 1998, now U.S. Pat. No. 6,421,704.

Parent Case Text (9):
U.S. patent application Ser. No. 09/044,919, entitled "Deferred Reconstruction of Objects and Remote Loading for Event Notification in a Distributed System, " filed on Mar. 20, 1998, now U.S. Pat. No. 6,272,559.

Parent Case Text (11):
U.S. patent application Ser. No. 09/044,790, entitled "Method and Apparatus for Determining Status of Remote Objects in a Distributed System," filed on Mar. 20, 1998.

Parent Case Text (12):
U.S. patent application Ser. No. 09/044,930, entitled "Downloadable Smart Proxies for Performing Processing Associated with a Remote Procedure Call in a Distributed System," filed on Mar. 20, 1998, now U.S. Pat. No. 6,393,497.

Parent Case Text (17):
U.S. patent application Ser. No. 09/044,931, entitled "Dynamic Lookup Service in a Distributed System," filed on Mar. 20, 1998, now U.S. Pat. No. 6,185,611.

Parent Case Text (18):
U.S. patent application Ser. No. 09/044,939, entitled "Apparatus and Method for Providing Downloadable Code for Use in Communicating with a Device in a Distributed System," filed on Mar. 20, 1998.

Parent Case Text (19):
U.S. patent application Ser. No. 09/044,826, entitled "Method and System for Facilitating Access to a Lookup Service," filed on Mar. 20, 1998.

Parent Case Text (20):
U.S. patent application Ser. No. 09/044,932, entitled "Apparatus and Method for Dynamically Verifying Information in a Distributed System," filed on Mar. 20, 1998.

Parent Case Text (21):
U.S. patent application Ser. No. 09/030,840, entitled "Method and Apparatus for Dynamic Distributed Computing Over a Network," and filed on Feb. 26, 1998.

Brief Summary Text (2):
The present invention relates to a system and method for transmitting objects between

machines in a distributed system and more particularly relates to methods for invocation of remote methods in a distributed system.

Brief Summary Text (4):
Distributed programs that concentrate on point-to-point data transmission can often be adequately and efficiently handled using special-purpose protocols for remote terminal access and file transfer. Such protocols are tailored to a specific program and, therefore, do not provide a foundation on which to build a variety of distributed programs (e.g., distributed operating systems, electronic mail systems, conferencing systems, etc.).

Brief Summary Text (5):
While conventional transport services can be used as the basis for building distributed programs, these services exhibit many organizational problems, such as the use of different data types in different machines, lack of facilities for synchronization, and no provision for a limited programming paradigm.

Brief Summary Text (6):
Distributed systems usually contain a number of different types of machines interconnected by communications networks. Each machine has its own internal data types, its own address alignment rules, and its own operating system. This heterogeneity causes problems when building distributed systems. As a result, program developers must include in programs developed for such heterogeneous distributed systems the capability of dealing with ensuring that information is handled and interpreted consistently on different machines.

Brief Summary Text (17):
Because the JVM may be implemented on any type of platform, implementing distributed programs using the JVM significantly reduces the difficulties associated with developing programs for heterogenous distributed systems. Moreover, the JVM uses a remote method invocation (RMI) system that enables communication among programs of the system. RMI is explained in, for example, the following document, which is incorporated herein by reference: Remote Method Invocation Specification, Sun Microsystems, Inc. (1997), which is available via universal resource locator (URL) www.javasoft.com/products/jdk/1.1/docs/guide/rmi/spec/rmiTOC.doc.html.

Brief Summary Text (18):
FIG. 2 is a diagram illustrating the flow of objects in an object-oriented distributed system 200 including machines 201 and 202 for transmitting and receiving method invocations using the JVM. In system 200, machine 201 uses RMI 205 for responding to a call from object 203 to invoke a method on remote object 204 by converting the call, including an identification of the method and any parameters into a byte stream 207. While machine 201 is responding to the call from object 203, a process running on the same or another machine in system 200 may continue operation without waiting for a response.

Brief Summary Text (19):
Machine 202 receives the byte stream 207. Using RMI 206, machine 202 automatically converts it into executable bytecode to initiate the invocation of the method on object 204. RMI can also be used to transport objects within the distributed system for use in connection with processes executing on remote machines.

Drawing Description Text (4):
FIG. 2 is a diagram illustrating the transmission of objects in an object-oriented distributed system;

Drawing Description Text (5):
FIG. 3 is a diagram of an exemplary distributed processing system that can be used in an implementation consistent with the present invention;

Drawing Description Text (6):
FIG. 4 is a diagram of an exemplary distributed system infrastructure;

Drawing Description Text (7):
FIG. 5 is a diagram of a computer in the distributed system infrastructure shown in FIG. 4;

Detailed Description Text (8):
Distributed Processing System

Detailed Description Text (9):
FIG. 3 illustrates a distributed processing system 300 which can be used in an implementation consistent with the present invention. In FIG. 3, distributed processing system 300 contains three independent and heterogeneous platforms 301, 302, and 303 connected in a network configuration represented by network cloud 319. The composition and protocol of the network configuration represented by cloud 319 is not important as long as it allows for communication of the information between platforms 301, 302 and 303. In addition, the use of just three platforms is merely for illustration and does not limit an implementation consistent with the present invention to the use of a particular number of platforms. Further, the specific network architecture is not crucial to embodiments consistent with this invention. For example, another network architecture that could be used in an implementation consistent with this invention would employ one platform as a network controller to which all the other platforms would be connected.

Detailed Description Text (10):
In the implementation of distributed processing system 300, platforms 301, 302 and 303 each include a processor 316, 317, and 318 respectively, and a memory, 304, 305, and 306, respectively. Included within each memory 304, 305, and 306, are applications 307, 308, and 309, respectively, operating systems 310, 311, and 312, respectively, and RMI components 313, 314, and 315, respectively.

Detailed Description Text (15):
Distributed System Infrastructure

Detailed Description Text (16):
Systems and methods consistent with the present invention may also operate within an exemplary particular distributed system 400, which will be described with reference to FIGS. 4 and 5. This distributed system 400 is comprised of various components, including hardware and software, to (1) allow users of the system to share services and resources over a network of many devices; (2) provide programmers with tools and programming patterns that allow development of robust, secured distributed systems; and (3) simplify the task of administering the distributed system. To accomplish these goals, distributed system 400 utilizes the Java programming environment to allow both code and data to be moved from device to device in a seamless manner. Accordingly, distributed system 400 is layered on top of the Java programming environment and exploits the characteristics of this environment, including the security offered by it and the strong typing provided by it.

Detailed Description Text (17):
In distributed system 400 of FIGS. 4 and 5, different computers and devices are federated into what appears to the user to be a single system. By appearing as a single system, distributed system 400 provides the simplicity of access and the power of sharing that can be provided by a single system without giving up the flexibility and personalized response of a personal computer or workstation. Distributed system 400 may contain thousands of devices operated by users who are geographically disperse, but who agree on basic notions of trust, administration, and policy.

Detailed Description Text (18):
Within an exemplary distributed system are various logical groupings of services provided by one or more devices, and each such logical grouping is known as a Djinn. A "service" refers to a resource, data, or functionality that can be accessed by a user, program, device, or another service and that can be computational, storage related, communication related, or related to providing access to another user. Examples of services provided as part of a Djinn include devices, such as printers, displays, and disks; software, such as programs or utilities; information, such as databases and files; and users of the system.

Detailed Description Text (20):
Distributed system 400 is comprised of computer 402, a computer 404, and a device 406 interconnected by a network 408. Device 406 may be any of a number of devices, such as a printer, fax machine, storage device, computer, or other devices. Network 408 may be a local area network, wide area network, or the Internet. Although only two computers and one device are depicted as comprising distributed system 400, one skilled in the art will appreciate that distributed system 400 may include additional computers or devices.

Detailed Description Text (21):

FIG. 5 depicts computer 402 in greater detail to show a number of the software components of distributed system 400. One skilled in the art will appreciate that computer 404 or device 406 may be similarly configured. Computer 402 includes a memory 502, a secondary storage device 504, a central processing unit (CPU) 506, an input device 508, and a video display 510. Memory 502 includes a lookup service 512, a discovery server 514, and a Java runtime system 516. The Java runtime system 516 includes the Java RMI system 518 and a JVM 520. Secondary storage device 504 includes a Java space 522.

Detailed Description Text (22):
As mentioned above, distributed system 400 is based on the Java programming environment and thus makes use of the Java runtime system 516. The Java runtime system 516 includes the Java API libraries, allowing programs running on top of the Java runtime system to access, in a platform-independent manner, various system functions, including windowing capabilities and networking capabilities of the host operating system. Since the Java API libraries provide a single common API across all operating systems to which the Java runtime system is ported, the programs running on top of a Java runtime system run in a platform-independent manner, regardless of the operating system or hardware configuration of the host platform. The Java runtime system 516 is provided as part of the Java software development kit available from Sun Microsystems, Inc. of Mountain View, Calif.

Detailed Description Text (24):
Lookup service 512 defines the services that are available for a particular Djinn. That is, there may be more than one Djinn and, consequently, more than one lookup service within distributed system 400. Lookup service 512 contains one object for each service within the Djinn, and each object contains various methods that facilitate access to the corresponding service. Lookup service 512 is described in U.S. patent application entitled "Method and System for Facilitating Access to a Lookup Service," which was previously incorporated herein by reference.

Detailed Description Text (25):
Discovery server 514 detects when a new device is added to distributed system 400, during a process known as boot and join (or discovery), and when such a new device is detected, the discovery server passes a reference to lookup service 512 to the new device so that the new device may register its services with the lookup service and become a member of the Djinn. After registration, the new device becomes a member of the Djinn, and as a result, it may access all the services contained in lookup service 512. The process of boot and join is described in U.S. patent application entitled "Apparatus and Method for providing Downloadable Code for Use in Communicating with a Device in a Distributed System," which was previously incorporated herein by reference.

Detailed Description Text (26):
A Java space 522 is an object repository used by programs within distributed system 400 to store objects. Programs use a Java space 522 to store objects persistently as well as to make them accessible to other devices within distributed system 400. Java spaces are described in U.S. patent application Ser. No. 08/971,529, entitled "Database System Employing Polymorphic Entry and Entry Matching," assigned to a common assignee, and filed on Nov. 17, 1997, which is incorporated herein by reference. One skilled in the art will appreciate that an exemplary distributed system 400 may contain many lookup services, discovery servers, and Java spaces.

Detailed Description Text (27):
Data Flow in a Distributed Processing System

Detailed Description Text (28):
FIG. 6 is a block diagram of an object-oriented distributed network 600 connecting machines 601 and 606, such as computers or virtual machines executing on one or more computers, or the machines described with reference to FIGS. 3, 4, and 5. Network 600 includes a client machine 601 containing RMI 602 and associated code 603. A server machine 606 includes RMI 605 and remote object 608. In operation, RMI 602 transmits a call or request 609 to RMI 605, requesting invocation of a method of remote object 608. RMI 602 uses a generic proxy 604 for transmitting call 609. Generic proxy 604 provides an advantage of not being type-specific so that it may invoke methods of varying types of remote objects. Table 1 contains a class definition written in the Java programming language for a generic proxy class by Netscape Communications Corp. and extended to allow the generic proxy to support invocation to methods among a set of interfaces.

Detailed Description Text (34):
Further details on the use of a method hash are disclosed in U.S. patent application entitled "Method and System for Deterministic Hashes to Identify Remote Methods," which was previously incorporated herein by reference. Marshalling involves constructing an object from a byte stream including code or a reference to code for use in the construction. Marshalling and unmarshalling are explained in U.S. patent application Ser. No. 08/950,756, filed on Oct. 15, 1997, and entitled "Deferred Reconstruction of Objects and Remote Loading in a Distributed System," now allowed, which is incorporated herein by reference.

Other Reference Publication (3):
Aldrich et al., "Providing Easier Access to Remote Objects in Distributed Systems," Calif. Institute of Technology, www.cs.caltech.edu/%7Ejedi/paper/jedipaper.html, Nov. 21, 1997.

Other Reference Publication (4):
Dave et al., "Proxies, Application Interfaces, and Distributed Systems," XP 002009478, IEEE, pp. 212-220, Sep. 1992.

Other Reference Publication (6):
Fleisch et al., "High Performance Distributed Objects Using Distributed Shared Memory & Remote Method Invocation," System Sciences, 1998, Proceedings of the 31st Hawaii Internat'l. Conference, Jan. 6-9, 1998, pp. 574-578.

Other Reference Publication (8):
Guyennet et al., "Distributed Shared Memory Layer for Cooperative Work Applications," IEEE, 0742-1303/97, pp. 72-78, 1997.

Other Reference Publication (17):
Betz, Mark, "Interoperable objects: laying the foundation for distributed object computing"; Dr. Dobb's Journal, vol. 19, No. 11, p. 18(13), Oct., 1994.

Other Reference Publication (18):
Bevan, D.I., "An Efficient Reference Counting Solution To The Distributed Garbage Collection Problem", Parallel Computing, NL, Elsevier Publishers, Amsterdam, vol. 9, No. 2, Jan. 9, 1989, pp. 179-192.

Other Reference Publication (20):
Dave A. et al., "Proxies, Application Interface, and Distributed Systems", Proceedings International Workshop on Object Orientation in Operating Systems, Sep. 24, 1992, pp. 212-220.

Other Reference Publication (27):
Guth, Rob: "JavaOne: Sun to Expand Java Distributed Computing Effort", "HTTP://WWW.SUNWORLD.COM/SWOL-02-1998/SWOL-02-SUNSPOTS.HTML," XP-002109935, 1998, p. 1.

Other Reference Publication (28):
Hamilton et al., "Subcontract: a flexible base for distributed programming"; Proceedings of 14th Symposium of Operating System Principles, Dec., 1993.

Other Reference Publication (33):
IBM: "Chapter 6--Distributed SOM (DSOM)" Somobjects Developer Toolkit Users Guide, Version 2.1, Oct., 1994 (1994-10), pp. 6-1-6-90.

Other Reference Publication (40):
Orfali R. et al., "The Essential Distributed Objects Survival Guide," Chapter 11: Corba Commercial ORBs, John Wiley & Sons, Inc., 1996, pp. 203-215.

Other Reference Publication (43):
Waldo J et al: "Events in an RPC based distributed system" Proceedings of the 1995 USENIX Technical Conference, Proceedings USENIX Winter 1995 Technical Conference, New Orleans, LA. USA, Jan. 16-20, 1995, pp. 131-142.

Other Reference Publication (46):
Yemini, Y. and S. da silva, "Towards Programmable Networks", IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, L'Aquila, Italy, Oct., 1996, pp. 1-11.

Other Reference Publication (48):
Mullender, Distributed Systems, Second Edition, Addison-Wesley, 1993.

Other Reference Publication (49):
Howard et al., Scale and Performance in a Distributed File System, ACM Transactions on Computer Systems, vol. 6, No. 1, Feb. 1988, pp. 51-81.

Other Reference Publication (51):
Dijkstra, Self-stabilizing Systems in Spite of Distributed Control, Communications of the ACM, vol. 17, No.11, Nov. 1974, pp. 643-644.

Other Reference Publication (54):
Sharrott et al., ObjectMap: Integrating High Performance Resources into a Distributed Object-oriented Environment, ICODP, 1995.

Other Reference Publication (55):
Birrell et al., Grapevine: An Exercise in Distributed Computing, Communications of the ACM, vol. 25, No. 4, Apr. 1982, pp. 260-274.

Other Reference Publication (57):
Gray et al., Leases: An Efficient Fault-Tolerant Mechanism for Distributed File Cache Consistency, ACM, 1989, pp. 202-210.

Other Reference Publication (59):
Dolev et al., On the Minimal Synchronism Needed for Distributed Consensus, Journal of the ACM, vol. 34, No. 1, Jan. 1987, pp. 77-97.

Other Reference Publication (60):
Mummert et al., Long Term Distributed File Reference Tracing: Implementation and Experience, Carnegie Mellon University School of Computer Science, Nov. 1994, pp. 1-28.

Other Reference Publication (66):
Carriero et al., Distributed Data Structures in Linda, Principals of Programming Language, 1986, pp. 1-16.

Other Reference Publication (69):
Carriero et al, Distributed Data Structures in Linda, Yale Research Report YALEU/DCS/RR-438, Nov. 1985.

Other Reference Publication (70):
Agha et al., Actorspaces: An Open Distributed Programming Paradigm, University of Illinois, Report No. UIUCDCS-R-92-1766, Open Systems Laboratory TR No. 8, Nov. 1992, pp. 1-12.

Other Reference Publication (72):
Liskov et al., Distributed Object Management in Thor, International Workshop on Distributed Object Management, 1992, pp. 12.

Other Reference Publication (73):
Coulouris et al., Distributed Systems Concepts and Designs, Second Edition, Addison-Wesley, 1994.

Other Reference Publication (75):
Birrell et al., Distributed Garbage Collection for Network Objects, DEC SRC Research Report 116, Dec. 15, 1993.

Other Reference Publication (77):
Wollrath et al., A Distributed Object Model for the Java.TM. System, USENIX Association, Conference on Object-Oriented Technologies and Systems, Jun. 17-21, 1996.

Other Reference Publication (78):
Harris et al., Proposal for a General Java Proxy Class for Distributed Systems and Other Uses, Netscape Communications Corp., Jun. 25, 1997.

Other Reference Publication (92):
Almes et al., "Research in Integrated Distributed Computing," Department of Computer Science, University of Washington, Oct. 1979, pp. 1-42.

Other Reference Publication (98):
Black et al., "A Language for Distributed Programming," Department of Computer Science, University of Washington, Technical Report 86-02-03, Feb. 1986, p. 10.

Other Reference Publication (102):
Black, "Supporting Distributed Applications: Experience with Eden," Department of Computer Science, University of Washington, Technical Report 85-03-02, Mar. 1985, pp. 1-21.

Other Reference Publication (106):
Burns et al., "An Analytical Study of Opportunistic Lease Renewal," Distributed Computing Systems, 21st International Conference, pp. 146-153, Apr. 2000.

Other Reference Publication (107):
Ciancarini et al., "Coordinating Distributed Applets with Shade/Java," Feb. 1998, pp. 130-138.

Other Reference Publication (114):
Hutchinson, "Emerald: An Object-Based Language for Distributed Programming,"a Dissertation, University of Washington, 1987, pp. 1-107.

Other Reference Publication (115):
Jacob, "The Use of Distributed Objects and Dynamic Interfaces in a Wide-Area Transaction Environment,"SIGCOMMn '95 Workshop on Middleware: Cambridge, Mass., Aug. 1995, pp. 1-3.

Other Reference Publication (117):
Jul, "Object Mobility in a Distributed Object-Oriented System," à Dissertation, University of Washington, 1989, pp. 1-154 (1 page Vita).

Other Reference Publication (118):
Koshizuka et al., "WIndow Real-Objects: A Distributed Shared Memory for Distributed Implementation of GUI Applications,", Nov. 1993, pp. 237-247.

Other Reference Publication (123):
Pu, "Replication and Nested Transaction in the Eden Distributed System," Doctoral Dissertation, University of Washington, Aug. 6, 1986, pp. 1-179 (1 page Vita).

Other Reference Publication (124):
Trehan et al., "Toolkit for Shared Hypermedia on a Distributed Object Oriented Architecture,"1993, pp. 1-8.

CLAIMS:

1. A method for remote method invocation in a distributed system comprised of multiple machines, comprising: receiving a request to invoke a method of an object; determining the method to be invoked using a generic code, wherein the generic code is not pregenerated; invoking the method based on the determination; and providing an indication of the invoked method.

3. A method for remote method invocation in a distributed object oriented system having multiple machines and defining a plurality of classes, comprising: receiving a request to invoke a method of an object of a specific class; determining the method using a common process for all of the classes wherein the common process is not pregenerated; invoking the method based on the determination; and providing an indication of the invoked method.

5. A method for remote method invocation in a distributed system comprised of multiple machines, comprising: receiving a request to invoke a first method of a first object and a second method of a second object being of a different type from the first object; determining the first and second methods to be invoked using a common process, wherein the common process is not pregenerated; invoking the first and second methods based on the determination; and providing an indication of the first invoked method and the second invoked method.

6. An apparatus for remote method invocation in a distributed system comprised of multiple machines, comprising: a module configured to receive a request to invoke a method of an object; a module configured to determine the method to be invoked using a generic code, wherein the generic code is not pregenerated; a module configured to

invoke the method based on the determination; and a module configured to provide an indication of the invoked method.

8. An apparatus for remote method invocation in a distributed system comprised of multiple machines, comprising: a module configured to receive a request to invoke a method of an object of a specific class; a module configured to determine the method using a common process for all of the classes, wherein the common process is not pregenerated; a module configured to invoke the method based on the determination; and a module configured to provide an indication of the invoked method.

10. An apparatus for remote method invocation in a distributed system comprised of multiple machines, comprising: a module configured to receive a request to invoke a first method of a first object and a second method of a second object being of a different type from the first object; a module configured to determine the first and second methods to be invoked using a common process, wherein the common process is not pregenerated; a module configured to invoke the first and second methods based on the determination; and a module configured to provide an indication of the first invoked method and the second invoked method.

11. A system for transmitting objects in a distributed system comprised of multiple machines, comprising: a first machine; a second machine; a network connecting the first machine with the second machine; and an apparatus for transmitting objects, the apparatus including: a module configured to receive a request to invoke a method of an object; a module configured to determine the method to be invoked using a generic code, wherein the generic code is not pregenerated; a module configured to invoke the method based on the determination; and a module configured to provide an indication of the invoked method.

13. A system for transmitting objects in a distributed system comprised of multiple machines, comprising: a first machine; a second machine; a network connecting the first machine with the second machine; and an apparatus for transmitting objects, the apparatus including: a module configured to receive a request to invoke a method of an object of a specific class; a module configured to determine the method using a common process for of the classes, wherein the common process is not pregenerated; a module configured to invoke the method based on the determination; and a module configured to provide an indication of the invoked method.

15. A system for transmitting objects in a distributed system comprised of multiple machines, comprising: a first machine; a second machine; a network connecting the first machine with the second machine; and an apparatus for transmitting objects, the apparatus including: a module configured to receive a request to invoke a first method of a first object and a second method of a second object being of a different type from the first object; a module configured to determine the first and second methods to be invoked using a common process, wherein the common process is not pregenerated; a module configured to invoke the first and second methods based on the determination; and a module configured to provide an indication of the first invoked method and the second invoked method.

21. An apparatus for remote method invocation in a distributed system comprised of multiple machines, comprising: means for receiving a request to invoke a method of an object; means for determining the method to be invoked using a generic code, wherein the generic code is not pregenerated; means for invoking the method based on the determination; and means for providing an indication of the invoked method.

22. A distributed system for remote method invocation comprising: a first module for requesting an invocation of a remote object based on a generic proxy; and a second module for executing generic code to invoke a remote object corresponding to the invocation request and returning an associated response to the first module.

23. The distributed system of claim 22, wherein the generic proxy is initialized to indicate a plurality of executable methods.

24. A distributed processing system comprising: a client machine having a remote procedure call module; a serve machine having a remote object with at least one method; a network communicably connecting the client machine and the server machine, wherein when a code executing on the client machine seeks to invoke the method of the remote object, the client remote procedure uses a generic proxy that lacks a specific type to transmit a request to the remote object.

WEST

Generate Collection

L13: Entry 11 of 90 File: USPT Oct 15, 2002

DOCUMENT-IDENTIFIER: US 6466947 B2
TITLE: Apparatus and method for dynamically verifying information in a distributed system

DATE FILED (1):
19980320

Abstract Text (1):
Use of a policy object for verification in a distributed system. A machine downloads a policy object containing a reference to code governing verification of data. The machine uses the reference to obtain the code and locally verify data or other information. As particular rules for the data change, the policy object may be updated to provide a reference to the code for the new rules when it is downloaded.

Parent Case Text (2):
Provisional U.S. patent application No. 60/076,048, now expired entitled "Distributed Computing System," filed on Feb. 26, 1998.

Parent Case Text (4):
U.S. patent application Ser. No. 09/044,838, entitled "Method, Apparatus, and Product for Leasing of Delegation Certificates in a Distributed System, " and filed on the same date herewith.

Parent Case Text (5):
U.S. patent application Ser. No. 09/044,834 now pending, entitled "Method, Apparatus and Product for Leasing of Group Membership in a Distributed System, " filed on the same date herewith.

Parent Case Text (8):
U.S. patent application Ser. No. 09/044,919 now U.S.Pat. No. 6,272,559, entitled "Deferred Reconstruction of Objects and Remote Loading for Event Notification in a Distributed System," and filed on the same date herewith.

Parent Case Text (11):
U.S. patent application Ser. No. 09/044,790 now pending, entitled "Method and Apparatus for Determining Status of Remote Objects in a Distributed System, " and filed on the same date herewith.

Parent Case Text (12):
U.S. patent application Ser. No. 09/044,930, now U.S. Pat. No. 6,393,497, entitled "Downloadable Smart Proxies for Performing Processing Associated with a Remote Procedure Call in a Distributed System," and filed on same date herewith.

Parent Case Text (17):
U.S. patent application Ser. No. 09/044,931 now U.S. Pat. No. 6,185,611, entitled "Dynamin Lookup Service in a Distributed System," and filed on the same date herewith.

Parent Case Text (18):
U.S. patent application Ser. No. 09/044,939 now pending, entitled "Apparatus and Method for Providing Downloadable Code for Use in Communicating with a Device in a Distributed System," filed on the same date herewith.

Parent Case Text (19):
U.S. patent application Ser. No. 09/044,826 now pending, entitled "Method and System for Facilitating Access to a Lookup Service," and filed on the same date herewith.

Brief Summary Text (2):

The present invention relates to a system and method for transmitting objects between machines in a distributed system and more particularly to dynamically verifying information in a distributed system.

Brief Summary Text (4):
Distributed programs which concentrate on point-to-point data transmission can often be adequately and efficiently handled using special-purpose protocols for remote terminal access and file transfer. Such protocols are tailored specifically to the one program and do not provide a foundation on which to build a variety of distributed programs (e.g., distributed operating systems, electronic mail systems, computer conferencing systems, etc.).

Brief Summary Text (5):
While conventional transport services can be used as the basis for building distributed programs, these services exhibit many organizational problems, such as the use of different data types in different machines, lack of facilities for synchronization, and no provision for a simple programming paradigm.

Brief Summary Text (6):
Distributed systems usually contain a number of different types of machines interconnected by communications networks. Each machine has its own internal data types, its own address alignment rules, and its own operating system. This heterogeneity causes problems when building distributed systems. As a result, program developers must include in programs developed for such heterogeneous distributed systems the capability of dealing with ensuring that information is handled and interpreted consistently on different machines.

Brief Summary Text (16):
Because the JVM may be implemented on any type of platform, implementing distributed programs using the JVM significantly reduces the difficulties associated with developing programs for heterogenous distributed systems. Moreover, the JVM uses a Java remote method invocation system (RMI) that enables communication among programs of the system. RMI is explained in, for example, the following document, which is incorporated herein by reference: Remote Method Invocation Specification, Sun Microsystems, Inc. (1997), which is available via universal resource locator (URL) http://www.javasoft.com/products/jdk/1.1/docs/guide/rmi/spec/rmiTOC.doc. html.

Brief Summary Text (17):
FIG. 2 is a diagram illustrating the flow of objects in an object-oriented distributed system 200 including machines 201 and 202 for transmitting and receiving method invocations using the JVM. In system 200, machine 201 uses RMI 205 for responding to a call for object 203 by converting the object into a byte stream 207 including an identification of the type of object transmitted and data constituting the object. While machine 201 is responding to the call for object 203, a process running on the same or another machine in system 200 may continue operation without waiting for a response to its request.

Brief Summary Text (19):
The communication among the machines may include verification of data or other information. Such verification typically requires multiple calls for verification of particular data or other information, which may result in a large volume of calls and potentially increased expense for the verification. Accordingly, a need exists for efficient verification of data or other information in a distributed system.

Drawing Description Text (4):
FIG. 2 is a diagram illustrating the transmission of objects in an object-oriented distributed system;

Drawing Description Text (5):
FIG. 3 is a diagram of an exemplary distributed processing system that can be used in an implementation consistent with the present invention;

Drawing Description Text (6):
FIG. 4 is a diagram of an exemplary distributed system infrastructure;

Drawing Description Text (7):
FIG. 5 is a diagram of a computer in a distributed system infrastructure shown in FIG. 4;

Drawing Description Text (8):
FIG. 6 is a diagram of an exemplary distributed network for use in transmission of a policy object; and

Detailed Description Text (3):
Machines consistent with the present invention may use a policy object, also referred to as a verification object, in a distributed system, the policy object performing processing when verification is needed. A machine downloads a policy object containing a reference to code governing verification of data or other information. The machine uses the reference to obtain the code and locally verify, for example, data constraints among items, data items, or objects. A verification object may also be used to verify other types of information. As particular rules for the data or information change, the policy object may be updated to provide a reference to the code for the new rules when it is downloaded.

Detailed Description Text (4):
Systems consistent with the present invention may efficiently transfer policy objects using a variant of an RPC or RMI, passing arguments and return values from one process to another process each of which may be on different machines. The term "machines" is used in this context to refer to a physical machine or a virtual machine. Multiple virtual machines may exist on the same physical machine. Examples of RPC systems include distributed computed environment (DCE) RPC and Microsoft distributed common object model (DCOM) RPC.

Detailed Description Text (5):
Distributed Processing Systems

Detailed Description Text (6):
FIG. 3 illustrates an exemplary distributed processing system 300 which can be used in an implementation consistent with the present invention. In FIG. 3, distributed processing system 300 contains three independent and heterogeneous platforms 301, 302, and 303 connected in a network configuration represented by network cloud 319. The composition and protocol of the network configuration represented by cloud 319 is not important as long as it allows for communication of the information between platforms 301, 302 and 303. In addition, the use of just three platforms is merely for illustration and does not limit an implementation consistent with the present invention to the use of a particular number of platforms. Further, the specific network architecture is not crucial to embodiments consistent with this invention. For example, another network architecture that could be used in an implementation consistent with this invention would employ one platform as a network controller to which all the other platforms would be connected.

Detailed Description Text (7):
In the implementation of distributed processing system 300, platforms 301, 302 and 303 each include a processor 316, 317, and 318 respectively, and a memory, 304, 305, and 306, respectively. Included within each memory 304, 305, and 306, are applications 307, 308, and 309, respectively, operating systems 310, 311, and 312, respectively, and RMI components 313, 314, and 315, respectively.

Detailed Description Text (12):
Distributed System Infrastructure

Detailed Description Text (13):
Systems and methods consistent with the present invention may also operate within a particular distributed system 400, which will be described with reference to FIGS. 4 and 5. This distributed system 400 is comprised of various components, including hardware and software, to (1) allow users of the system to share services and resources over a network of many devices; (2) provide programmers with tools and programming patterns that allow development of robust, secured distributed systems; and (3) simplify the task of administering the distributed system. To accomplish these goals, distributed system 400 utilizes the Java programming environment to allow both code and data to be moved from device to device in a seamless manner. Accordingly, distributed system 400 is layered on top of the Java programming environment and exploits the characteristics of this environment, including the security offered by it and the strong typing provided by it.

Detailed Description Text (14):
In distributed system 400 of FIGS. 4 and 5, different computers and devices are federated into what appears to the user to be a single system. By appearing as a single

system, distributed system 400 provides the simplicity of access and the power of sharing that can be provided by a single system without giving up the flexibility and personalized response of a personal computer or workstation. Distributed system 400 may contain thousands of devices operated by users who are geographically disperse, but who agree on basic notions of trust, administration, and policy.

Detailed Description Text (15):
Within an exemplary distributed system are various logical groupings of services provided by one or more devices, and each such logical grouping is known as a Djinn. A "service" refers to a resource, data, or functionality that can be accessed by a user, program, device, or another service and that can be computational, storage related, communication related, or related to providing access to another user. Examples of services provided as part of a Djinn include devices, such as printers, displays, and disks; software, such as programs or utilities; information, such as databases and files; and users of the system.

Detailed Description Text (17):
Distributed system 400 is comprised of computer 402, a computer 404, and a device 406 interconnected by a network 408. Device 406 may be any of a number of devices, such as a printer, fax machine, storage device, computer, or other devices. Network 408 may be a local area network, wide area network, or the Internet. Although only two computers and one device are depicted as comprising distributed system 400, one skilled in the art will appreciate that distributed system 400 may include additional computers or devices.

Detailed Description Text (18):
FIG. 5 depicts computer 402 in greater detail to show a number of the software components of distributed system 400. One skilled in the art will appreciate that computer 404 or device 406 may be similarly configured. Computer 402 includes a memory 502, a secondary storage device 504, a central processing unit (CPU) 506, an input device 508, and a video display 510. Memory 502 includes a lookup service 512, a discovery server 514, and a Java runtime system 516. The Java runtime system 516 includes the Java RMI system 518 and a JVM 520. Secondary storage device 504 includes a Java space 522.

Detailed Description Text (19):
As mentioned above, distributed system 400 is based on the Java programming environment and thus makes use of the Java runtime system 516. The Java runtime system 516 includes the Java API libraries, allowing programs running on top of the Java runtime system to access, in a platform-independent manner, various system functions, including windowing capabilities and networking capabilities of the host operating system. Since the Java API libraries provides a single common API across all operating systems to which the Java runtime system is ported, the programs running on top of a Java runtime system run in a platform-independent manner, regardless of the operating system or hardware configuration of the host platform. The Java runtime system 516 is provided as part of the Java software development kit available from Sun Microsystems, Inc. of Mountain View, Calif.

Detailed Description Text (21):
Lookup service 512 defines the services that are available for a particular Djinn. That is, there may be more than one Djinn and, consequently, more than one lookup service within distributed system 400. Lookup service 512 contains one object for each service within the Djinn, and each object contains various methods that facilitate access to the corresponding service. Lookup service 512 is described in U.S. patent application entitled "Method and System for Facilitating Access to a Lookup Service," which was previously incorporated herein by reference.

Detailed Description Text (22):
Discovery server 514 detects when a new device is added to distributed system 400, during a process known as boot and join (or discovery), and when such a new device is detected, the discovery server passes a reference to lookup service 512 to the new device so that the new device may register its services with the lookup service and become a member of the Djinn. After registration, the new device becomes a member of the Djinn, and as a result, it may access all the services contained in lookup service 512. The process of boot and join is described in U.S. patent application entitled "Apparatus and Method for providing Downloadable Code for Use in Communicating with a Device in a Distributed System," which was previously incorporated herein by reference.

Detailed Description Text (23):
A Java space 522 is an object repository used by programs within distributed system 400 to store objects. Programs use a Java space 522 to store objects persistently as well as to make them accessible to other devices within distributed system 400. Java spaces are described in U.S. patent application Ser. No. 08/971,529, now U.S. Pat. No. 6,032,151 entitled "Database System Employing Polymorphic Entry and Entry Matching," assigned to a common assignee, and filed on Nov. 17, 1997, which is incorporated herein by reference. One skilled in the art will appreciate that an exemplary distributed system 400 may contain many lookup services, discovery servers, and Java spaces.

Detailed Description Text (24):
Data Flow in a Distributed Processing System

Detailed Description Text (25):
FIG. 6 is a diagram of an object-oriented distributed network 600 for use in transmission of a policy object for use in verification. Network 600 includes client machine 601 and server machine 604, which may be implemented with computers or virtual machines executing on one or more computers, or the machines described with reference to FIGS. 3, 4, and 5. Client machine 601 includes RMI 602 and associated object 603. Server machine 604 includes RMI 605 and associated policy object 606.

Detailed Description Text (30):
FIG. 7 is a flow chart of a process 700 for verification using a policy object, also referred to as a verification object. A machine first determines if verification is requested (step 701). If so, it requests a policy object from a server (step 702) and receives the policy object including a reference to code for use in verification of data or other information (step 703). Using the reference, it downloads code for the verification (step 704). The downloading of code may occur using the methods described in U.S. patent application Ser. No. 08/950,756, filed on Oct. 15, 1997, and entitled "Deferred Reconstruction of Objects and Remote Loading in a Distributed System," which is incorporated herein by reference.

Detailed Description Text (32):
Machines implementing the steps shown in FIG. 7 may include computer processors for performing the functions, as shown in FIGS. 3, 4, 5, and 6. They may include modules or programs configured to cause the processors to perform the above functions. They may also include computer program products stored in a memory. The computer program products may include a computer-readable medium or media having computer-readable code embodied therein for causing the machines to perform functions described in this specification. The media may include a computer data signal embodied in a carrier wave and representing sequences of instructions which, when executed by a processor, cause the processor to securely address a peripheral device at an absolute address by performing the method described in this specification. The media may also include data structures for use in performing the method described in this specification. In addition, the processing shown in FIG. 7 may occur through the use of smart proxies, which are described in U.S. patent application filed on even date herewith, assigned to a common assignee, and entitled "Downloadable Smart Proxies for Performing Processing Associated with a Remote Procedure Call in a Distributed System," which is incorporated herein by reference.

Other Reference Publication (2):
Aldrich et al., "Providing Easier Access to Remote Objects in Distributed Systems," Calif. Institute of Technology, www.cs.caltech.edu/%7Ejedi/paper/jedipaper.html, Nov. 21, 1997.

Other Reference Publication (3):
Burns et al., "An Analytical Study of Opportunistic Lease Renewal," Distributed Computing Systems, 21st International Conference, pp. 146-153, Apr. 2000.

Other Reference Publication (5):
Fleisch et al., "High Performance Distributed Objects Using Distributed Shared Memory & Remote Method Invocation," System Sciences, 1998, Proceedings of the 31st Hawaii Internat'l. Conference, Jan. 6-9, 1998, pp. 574-578.

Other Reference Publication (7):
Guyennet et al., "Distributed Share Memory Layer for Cooperative Work Applications," IEEE, 0742-1303/97, pp. 72-78, 1997.

Other Reference Publication (15):

Mullender, Distributed Systems, Second Edition, Addison-Wesley, 1993.

Other Reference Publication (16):
Howard et al., Scale and Performance in a Distributed File System, ACM Transactions on Computer Systems, vol. 6, No. 1, Feb. 1988, pp. 51-81.

Other Reference Publication (18):
Dijkstra, Self-stabilizing Systems in Spite of Distributed Control, Communications of the ACM, vol. 17, No. 11, Nov. 1974, pp. 643-644.

Other Reference Publication (21):
Sharrott et al., ObjectMap: Integrating High Performance Resources into a Distributed Object-oriented Environment, ICODP, 1995.

Other Reference Publication (22):
Birrell et al., Grapevine: An Exercise in Distributed Computing, Communications of the ACM, vol. 25, No. 4, Apr. 1982, pp. 260-274.

Other Reference Publication (24):
Gray et al., Leases: An Efficient Fault-Tolerant Mechanism for Distributed File Cache Consistency, ACM, 1989, pp. 202-210.

Other Reference Publication (26):
Dolev et al., On the Minimal Synchronism Needed for Distributed Consensus, Journal of the ACM, vol. 34, No. 1, Jan. 1987, pp. 77-97.

Other Reference Publication (27):
Mummert et al., Long Term Distributed File Reference Tracing: Implementation and Experience, Carnegie Mellon University School of Computer Science, Nov. 1994, pp. 1-28.

Other Reference Publication (33):
Carriero et al., Distributed Data Structures in Linda, Principals of Programming Language, 1986, pp. 1-16.

Other Reference Publication (36):
Carriero et al, Distributed Data Structures in Linda, Yale Research Report YALEU/DCS/RR-438, Nov. 1985.

Other Reference Publication (37):
Agha et al., Actorspaces: An Open Distributed Programming Paradigm, University of Illinois, Report No. UTUCDCS-R-92-1766, Open Systems Laboratory TR No. 8, Nov. 1992, pp. 1-12.

Other Reference Publication (39):
Liskov et al., Distributed Object Management in Thor, International Workshop on Distributed Object Management, 1992, pp. 12.

Other Reference Publication (40):
Coulouris et al., Distributed Systems Concepts and Designs, Second Edition, Addison-Wesley, 1994.

Other Reference Publication (42):
Birrell et al., Distributed Garbage Collection for Network Objects, DEC SRC Research Report 116, Dec. 15, 1993.

Other Reference Publication (44):
Wollrath et al., A Distributed Object Model for the Java.TM. System, USENIX Association, Conference on Object-Oriented Technologies and Systems, Jun. 17-21, 1996.

Other Reference Publication (45):
Harris et al., Proposal for a General Java Proxy Class for Distributed Systems and Other Uses, Netscape Communications Corp., Jun. 25, 1997.

Other Reference Publication (63):
Betz, Mark; "Interoperable objects: laying the foundation for distributed object computing"; Dr. Dobb's Journal, vol. 19, No. 11, p. 18(13); (Oct. 1994).

Other Reference Publication (64):

Bevan, D.I., "An Efficient Reference Counting Solution To The Distributed Garbage Collection Problem", Parall Computing, NL, Elsevier Publishers, Amsterdam, vol. 9, No. 2, pp. 179-192.

Other Reference Publication (66):
Dave A et al: "Proxies, Application Interface, and Distributed Systems", Proceedings International Workshop on Object Orientation in Operating Systems, Sep. 24, 1992, pp. 212-220.

Other Reference Publication (73):
Guth, Rob: "JavaOne: Sun to Expand Java Distributed Computing Effort", "HTTP://WWW.SUNWORLD.COM/SWOL-02-1998/SWOL-02-SUNSPOTS.HTML," XP-002109935, P.1, 1998.

Other Reference Publication (74):
Hamilton et al., "Subcontract: a flexible base for distributed programming"; Proceedings of 14th Symposium of Operating System Principles; (Dec. 1993).

Other Reference Publication (79):
IBM: "Chapter 6--Distributed SOM (DSOM)" Somobjects Developer Toolkit Users Guide, Version 2.1, Oct. 1994 (1994-10), pp. 6-1-6-90.

Other Reference Publication (85):
Orfali R. et al., "The Essential Distributed Objects Survival Guide," Chapter 11: Corba Commercial ORBs, pp. 203-215, John Wiley & Sons, Inc., (1996).

Other Reference Publication (89):
Waldo J et al: "Events in an RPC based distributed system" Proceedings of the 1995 Usenix Technical Conference, Proceedings Usenix Winter 1995 Technical Conference, New Orleans, LA. USA, Jan. 16-20, 1995, pp. 131-142.

Other Reference Publication (92):
Yemini, Y. and S. da silva, "Towards Programmable Networks", IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, L'Aquila, Italy, Oct. 1996.

CLAIMS:

1. A method for verifying data in a distributed system, comprising: determining whether a need exists to verify data; transmitting a request for a verification object; receiving from a romote device the verification object in response to the request, the verification object including a first executable code; constructing second executable code from the first executable code included in the verification object; and verifying the data by executing at least one of the first executable code included in the verification object, the second executable code, a combination of part of the first executable code included in the verification object and the second executable code, and a combination of both the first executable code included in the verification object and the second executable code.

5. A method for verifying data in a distributed system, comprising: determining whether a need exists to verify data; transmitting a request for a verification object; receiving from a remote device a response to the request including a first executable code; and constructing the verification object using the first executable code, the verification object exhibiting second executable code for processing associated with verifying the data.

21. A computer-readable medium containing instructions for causing a processor to perform a method for verifying data in a distributed system, the method comprising: determining whether a need exists to verify data; transmitting a request for a verification object; receiving from a remote device the verification object in response to the request, the verification object including a first executable code; constructing second executable code from the first executable code included in the verification object; and verifying the data by executing at least one of the first executable code included in the verification object, the second executable code, a combination of part of the first executable code included in the verification object and the second executable code, and a combination of both the first executable code included in the verification object and the second executable code.

25. A computer-readable medium containing instructions for causing a processor to perform a method for verifying data in a distributed system, the method comprising:

determining whether a need exists to verify data; transmitting a request for a verification object; receiving from a remote device a response to the request including a first executable code; and constructing the verification object using the first executable code, the verification object exhibiting second executable code for processing associated with verifying the data.

WEST

Generate Collection

L13: Entry 13 of 90 File: USPT Oct 8, 2002

DOCUMENT-IDENTIFIER: US 6463446 B1
TITLE: Method and apparatus for transporting behavior in an event-based distributed system

DATE FILED (1):
19980320

Abstract Text (1):
In a distributed computing system, a first process may register interest in an event occurring in another address space or physical machine in such a way as to allow the subsequent notification of the event's occurrence to contain an object that includes methods that are to be run on receipt of the notification. When the notification is received, either by the first process or by some other entity designated by the first process to be the final point of notification, the methods may be executed as specified by the first process.

Parent Case Text (3):
Provisional U.S. patent application Ser. No. 60/076,048, entitled "Distributed Computing System," filed on Feb. 26, 1998.

Parent Case Text (5):
U.S. patent application Ser. No. 09/044,838, entitled "Method, Apparatus, and Product for Leasing of Delegation Certificates in a Distributed System," and filed on Mar. 20, 1998.

Parent Case Text (6):
U.S. patent application Ser. No. 09/044,834, entitled "Method, Apparatus and Product for Leasing of Group Membership in a Distributed System," and filed on Mar. 20, 1998.

Parent Case Text (8):
U.S. patent application Ser. No. 09/044,919, entitled "Deferred Reconstruction of Objects and Remote Loading for Event Notification in a Distributed System," and filed on Mar. 20, 1998.

Parent Case Text (11):
U.S. patent application Ser. No. 09/044,790, entitled "Method and Apparatus for Determining Status of Remote Objects in a Distributed System," and filed on Mar. 20, 1998.

Parent Case Text (12):
U.S. patent application Ser. No. 09/044,930, entitled "Downloadable Smart Proxies for Performing Processing Associated with a Remote Procedure Call in a Distributed System," and filed on Mar. 20, 1998.

Parent Case Text (17):
U.S. patent application Ser. No. 09,044,931, entitled "Dynamic Lookup Service in a Distributed System," and filed on Mar. 20, 1998.

Parent Case Text (18):
U.S. patent application Ser. No. 09,044,939, entitled "Apparatus and Method for Providing Downloadable Code for Use in Communicating with a Device in a Distributed System," and filed on Mar. 20, 1998.

Parent Case Text (19):
U.S. patent application Ser. No. 09,044,826, entitled "Method and System for Facilitating Access to a Lookup Service," and filed on Mar. 20, 1998.

Parent Case Text (20):
U.S. patent application Ser. No. 09,044,932, entitled "Apparatus and Method for Dynamically Verifying Information in a Distributed System," and filed on Mar. 20, 1998.

Parent Case Text (21):
U.S. patent application Ser. No. 09/030,840, entitled "Method and Apparatus for Dynamic Distributed Computing Over a Network," and filed on Feb. 26, 1998.

Brief Summary Text (2):
This invention relates generally to a distributed computer system and more specifically to event handling procedures in a distributed computer system.

Brief Summary Text (4):
Distributed computer systems are systems in which the programming and data that the computer operates on are spread out over more than one computer, usually over a network with many computers. One conventional method for organizing a distributed system uses the client-server model, in which one part of the distributed system (the client) asks for some service from another part of the distributed system (the server). The server responds and then handles the client's request appropriately.

Brief Summary Text (6):
An alternative method for organizing distributed systems uses the event/notification model. In this model, a "listener" process, which is interested in the occurrence of an event within another entity within the system, registers its interest with a second process, a "notifier" process, designed to monitor such events. When the event occurs, the notifier notifies the listener or notifies another process (i.e., a third party process) designated by the listener when registering with the notifier. After receiving the event notification, that notified entity (i.e., the listener or the third party listener, as the case may be) may execute a function designated in the notification and present in the address space of the notified process. In the event-notification model, there is no single point of control as in the client-server model; instead, control is determined by notifications sent in response to the occurrence of designated events.

Brief Summary Text (7):
In a non-distributed system, the association of an event with the function to be run in response to the notification, called a callback function, is straightforward, since the event, the listener, and the callback function are all in a single address space. Life is not so simple in a distributed system. The listener may be in one address space or physical machine, the event may be generated in a second address space or physical machine (e.g., the address space of the notifier or an address space that the notifier monitors), and the notification may be sent to a third address space or physical machine (e.g., the address space of the third party process). This makes it difficult for the listener to insure that the third party process will be able to respond to the event in an appropriate manner, as the two may be separated and, possibly, might not even know of the other's existence. For example, the listener, to ensure that the correct function is executed by the third party, must keep track of whether the function is available in the third party's address space. This can be particularly burdensome when an administrator desires to update functions in the third party's address space, as all the potential distributed users of the new function must be updated on the status of the new function.

Brief Summary Text (8):
Thus, there is a need in the art to more effectively handle distributed programs collaborating using the event-notification model. This need is particularly poignant in systems in which the requesting entity is not able to easily verify the status of a function present at the executing entity.

Brief Summary Text (11):
To achieve the objects and in accordance with the purpose of the invention, as embodied and broadly described herein, a first aspect consistent with the present invention includes a method for controlling program execution in a distributed computer system comprising the steps of: (1) registering interest in an occurrence of an event in the distributed computer system, the registration of interest including information identifying the occurrence of the event, an identifier of a software entity in the distributed system, and a first object including a process and parameter data corresponding to the process; (2) monitoring at least a portion of the distributed computer system for the occurrence of the registered event; (3) notifying the software entity identified in the registration of interest when the event occurs, the

notification including a copy of the first object and an identification of the event that occurred; and (4) executing methods contained within the first object in response to the notifying step.

Brief Summary Text (12):
To achieve the objects and in accordance with the purpose of the invention, as embodied and broadly described herein, a second aspect consistent with the present invention includes a protocol for controlling the execution of processes in a distributed computer system, the protocol comprises a number of steps, including: (1) receiving a registration of interest in an event that is expected to occur in the distributed computer system, the registration including an identifier of a instructions for performing a process and parameter data corresponding to the process; (2) monitoring the distributed system for the occurrence of the registered event; and (3) notifying the software entity identified in the registration of interest when the event occurs, the notification including a copy of the first object and an identification of the event that occurred.

Drawing Description Text (3):
FIG. 1 is a diagram of an exemplary distributed system;

Drawing Description Text (4):
FIG. 2 is a diagram of an exemplary computer within the exemplary distributed system;

Drawing Description Text (5):
FIG. 3A is a flow chart illustrating methods consistent with the present invention for notifying and transporting behavior in an event based distributed system;

Drawing Description Text (7):
FIG. 4 is a block diagram illustrating a distributed computing system containing three exemplary computer platforms connected to one another via a network.

Detailed Description Text (2):
This disclosure describes a protocol allowing a listener process to register interest in an event occurring in another address space or physical machine in such a way as to allow the subsequent notification of the event's occurrence to contain an object that may include methods that are to be run on receipt of the notification. When the notification is received, either by the listener or by a third party, the methods may be executed as specified by the listener.

Detailed Description Text (4):
Methods and systems consistent with the present invention operate in a distributed system ("the exemplary distributed system") with various components, including both hardware and software. The exemplary distributed system (1) allows users of the system to share services and resources over a network of many devices; (2) provides programmers with tools and programming patterns that allow development of robust, secured distributed systems; and (3) simplifies the task of administering the distributed system. To accomplish these goals, the exemplary distributed system utilizes the Java.TM. programming environment to allow both code and data to be moved from device to device in a seamless manner. Accordingly, the exemplary distributed system is layered on top of the Java programming environment and exploits the characteristics of this environment, including the security offered by it and the strong typing provided by it. The Java programming environment is more clearly described in Jaworski, Java 1.1 Developer's Guide, Sams.net (1997), which is incorporated herein by reference.

Detailed Description Text (5):
In the exemplary distributed system, different computers and devices are federated into what appears to the user to be a single system. By appearing as a single system, the exemplary distributed system provides the simplicity of access and the power of sharing that can be provided by a single system without giving up the flexibility and personalized response of a personal computer or workstation. The exemplary distributed system may contain thousands of devices operated by users who are geographically disperse, but who agree on basic notions of trust, administration, and policy.

Detailed Description Text (6):
Within the exemplary distributed system are various logical groupings of services provided by one or more devices, and each such logical grouping is known as a Djinn. A "service" refers to a resource, data, or functionality that can be accessed by a user, program, device, or another service and that can be computational, storage related,

communication related, or related to providing access to another user. Examples of services provided as part of a Djinn include devices, such as printers, displays, and disks; software, such as applications or utilities; information, such as databases and files; and users of the system.

Detailed Description Text (8):
FIG. 1 depicts the exemplary distributed system 100 containing a computer 102, a computer 104, and a device 106 interconnected by a network 108. The device 106 may be any of a number of devices, such as a printer, fax machine, storage device, computer, or other devices. The network 108 may be a local area network, wide area network, or the Internet. Although only two computers and one device are depicted as comprising the exemplary distributed system 100, one skilled in the art will appreciate that the exemplary distributed system 100 may include additional computers or devices.

Detailed Description Text (9):
FIG. 2 depicts the computer 102 in greater detail to show a number of the software components of the exemplary distributed system 100. One skilled in the art will appreciate that computer 104 or device 106 may be similarly configured. Computer 102 includes a memory 202, a secondary storage device 204, a central processing unit (CPU) 206, an input device 208, and a video display 210. The memory 202 includes a lookup service 212, a discovery server 214, and a Java.TM. runtime system 216. The Java runtime system 216 includes the Java.TM. remote method invocation system (RMI) 218 and a Java.TM. virtual machine 220. The secondary storage device 204 includes a JavaSpace.TM. 222.

Detailed Description Text (10):
As mentioned above, the exemplary distributed system 100 is based on the Java programming environment and thus makes use of the Java runtime system 216. The Java runtime system 216 includes the Java.TM. API, allowing programs running on top of the Java runtime system to access, in a platform-independent manner, various system functions, including windowing capabilities and networking capabilities of the host operating system. Since the Java API provides a single common API across all operating systems to which the Java runtime system 216 is ported, the programs running on top of a Java runtime system run in a platform-independent manner, regardless of the operating system or hardware configuration of the host platform. The Java runtime system 216 is provided as part of the Java.TM. software development kit available from Sun Microsystems of Mountain View, Calif.

Detailed Description Text (12):
The lookup service 212 defines the services that are available for a particular Djinn. That is, there may be more than one Djinn and, consequently, more than one lookup service within the exemplary distributed system 100. The lookup service 212 contains one object for each service within the Djinn, and each object contains various methods that facilitate access to the corresponding service. The lookup service 212 and its access are described in greater detail in co-pending U.S. patent application Ser. No. 09,044,826, entitled "Method and System for Facilitating Access to a Lookup Service," which has previously been incorporated by reference.

Detailed Description Text (13):
The discovery server 214 detects when a new device is added to the exemplary distributed system 100, during a process known as boot and join or discovery, and when such a new device is detected, the discovery server passes a reference to the lookup service 212 to the new device, so that the new device may register its services with the lookup service and become a member of the Djinn. After registration, the new device becomes a member of the Djinn, and as a result, it may access all the services contained in the lookup service 212. The process of boot and join is described in greater detail in co-pending U.S. patent application Ser. No. 09/044,939, entitled "Apparatus and Method for providing Downloadable Code for Use in Communicating with a Device in a Distributed System," which has previously been incorporated by reference.

Detailed Description Text (14):
The JavaSpace 222 is an object repository used by programs within the exemplary distributed system 100 to store objects. Programs use the JavaSpace 222 to store objects persistently as well as to make them accessible to other devices within the exemplary distributed system. JavaSpaces are described in greater detail in co-pending U.S. patent application Ser. No. 08/971,529, entitled "Database System Employing Polymorphic Entry and Entry Matching," assigned to a common assignee, filed on Nov. 17, 1997, which is incorporated herein by reference. One skilled in the art will appreciate that the exemplary distributed system 100 may contain many lookup services, discovery

servers, and JavaSpaces.

Detailed Description Text (15):
Although systems and methods consistent with the present invention are described as operating in the exemplary distributed system and the Java programming environment, one skilled in the art will appreciate that the present invention can be practiced in other systems and other programming environments. Additionally, although aspects of the present invention are described as being stored in memory, one skilled in the art will appreciate that these aspects can also be stored on or read from other types of computer-readable media, such as secondary storage devices, like hard disks, floppy disks, or CD-ROM; a carrier wave from the Internet; or other forms of RAM or ROM. Sun, Sun Microsystems, the SunLogo, Java, and Java-based trademarks are trademarks or registered trademarks of Sun Microsystems Inc. in the United States and other countries.

Detailed Description Text (16):
FIG. 3A is a flow chart illustrating methods consistent with the present invention for notifying and transporting behavior in an event-based distributed system. Listener objects interested in the occurrence of a certain event must register an interest in the event. In particular, for each interest to be registered, the registering object sends a message to a notifier process, which is preferably an object containing procedures for monitoring the occurrence of the event (step 301). Generally, the notifier will be present in the virtual machine in which the event is expected to occur. The registration message includes: (1) implicit or explicit information identifying the event that is to be monitored, (2) information identifying the object that is to be notified when the event occurs, and (3) an object, or a reference to an object, that is to be passed to the notified object when the event occurs. The notifier may use a table to store information relating to registration messages it has received, such as the table shown in FIG. 3B. The notifier monitors its system or the network for the occurrence of the event (step 302). When the event occurs (step 303), the notifier notifies the object identified by the registering object that the registered event has occurred (step 304). Consistent with the present invention, the notification includes an identification of the event that occurred and the object or reference to the object that was passed to the notified object when the event occurred (step 304).

Detailed Description Text (18):
FIG. 4 is a block diagram illustrating a distributed computing system containing three exemplary computer systems 412, 413, and 414, connected to one another via a network. Computer systems 412-414 are physically similar to computer systems 102 and 112. In particular, each of computer systems 412, 413, and 414 includes a computer platform or machine 409, 410, and 411, respectively, executing a virtual machine 406, 407, or 408. Processes 402, 403, and 404 reside on their respective virtual machines 406, 407, and 408. Consistent with the present invention, computer systems 412, 413, and 414 form a distributed computing system using the event/notification model.

Detailed Description Text (19):
A hypothetical situation illustrating an application of the present invention will now be described with reference to the distributed system shown in FIG. 4. Assume that "administrator" process 402, executing in virtual machine 406, has the responsibility of monitoring the network and reporting potential problems to a human operator--such as a disk drive running out of free space. Notifier process 403, executing on virtual machine 407, monitors the system for the occurrence of disk drive full events. To register interest in a "DiskFull" event, process 402 transmits a registration message, called, for example, "RegisterInterest," to notify process 403, such as the message:

Detailed Description Text (23):
Because the SendPage object includes, within itself, all required methods and data for execution, SendPage is considered to be an object having "closure." By passing an object having closure to a designated entity, the object initially registering the object does not have to be aware of the functions available at the final destination of the passed object. This is particularly advantageous as it allows the object initially registering the event to easily modify the functionality of the passed object. For example, if the policy (functionality) of the SendPage object is created by an operator at computer 412, it is relatively easy for the operator to modify the policy of SendPage by, for example, instructing SendPage to page a backup technician if the page is not returned by the primary technician within twenty minutes.

Other Reference Publication (6):
Betz, Mark; "Interoperable objects: laying the foundation for distributed object

computing"; Dr. Dobb's Journal, vol. 19, No. 11, p. 18(13); (Oct. 1994).

Other Reference Publication (7):
Bevan, D.I., "An Efficient Reference Counting Solution To The Distributed Garbage Collection Problem", Parall Computing, NL, Elsevier Publishers, Amsterdam, vol. 9, No. 2, pp. 179-192.

Other Reference Publication (8):
Dave A et al: "Proxies, Applications Interface, and Distributed Systems", Proceedings International Workshop on Object Orientation in Operating Systems, Sep. 24, 1992, pp. 212-220.

Other Reference Publication (15):
Guth, Rob: "JavaOne: Sun to Expand Java Distributed Computing Effort", "Http://www.Sunworld.com/SWOL-02-1998/SWOL-02-Sunspots.Html," XP-002109935, p. 1, 1998.

Other Reference Publication (16):
Hamilton et al., "Subcontract: a flexible base for distributed programming"; Proceedings of 14th Symposium of Operating System Principles; (Dec. 1993).

Other Reference Publication (21):
IBM: "Chapter 6--Distributed SOM (DSOM)" SOMOObjects Developer Toolkit Users Guide, Version 2.1, (Oct. 1994), pp. 6-1-6-90.

Other Reference Publication (27):
Orfali R. et al., "The Essential Distributed Objects Survival Guide," Chapter 11: Corba Commercial ORBs, pp. 203-215, John Wiley & Sons, Inc., (1996).

Other Reference Publication (31):
Waldo J et al: "Events in an RFC based distributed system" Proceedings of the 1995 USENIX Technical Conference Proceedings USENIX Winter 1995 Technical Conference, New Orleans, LA, USA, Jan. 16-20, 1995, pp. 131-142.

Other Reference Publication (34):
Yemini, Y. and S. da silva, "Towards Programmable Networks", IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, L'Aquila, Italy, 10/96.

Other Reference Publication (38):
Mullender, Distributed Systems, Second Edition, Addison-Wesley, 1993.

Other Reference Publication (39):
Howard et al., Scale and Performance in a Distributed File System, ACM Transactions on Computer Systems, vol. 6, No. 1, Feb. 1988, pp. 51-81.

Other Reference Publication (41):
Dijkstra, Self-stabilizing Systems in Spite of Distributed Control, Communications of the ACM, vol. 17, No. 11, Nov. 1974, pp. 643-644.

Other Reference Publication (44):
Sharrott et al., ObjectMap: Integrating High Performance Resources into a Distributed Object-oriented Environment, ICODP, 1995.

Other Reference Publication (45):
Birrell et al., Grapevine: An Exercise in Distributed Computing, Communications of the ACM, vol. 25, No. 4, Apr. 1982, pp. 260-274.

Other Reference Publication (47):
Gray et al., Leases: An Efficient Fault-Tolerant Mechanism for Distributed File Cache Consistency, ACM, 1989, pp. 202-210.

Other Reference Publication (49):
Dolev et al., On the Minimal Synchronism Needed for Distributed Consensus, Journal of the ACM, vol. 34, No. 1, Jan. 1987, pp. 77-97.

Other Reference Publication (50):
Mummert et al., Long Term Distributed File Reference Tracing: Implementation and Experience, Carnegie Mellon University School of Computer Science, Nov. 1994, pp. 1-28.

Other Reference Publication (56):
Carriero et al., Distributed Data Structures in Linda, Principals of Programming Language, 1986, pp. 1-16.

Other Reference Publication (59):
Carriero et al, Distributed Data Structures in Linda, Yale Research Report YaleU/DCS/RR-438. Nov. 1985.

Other Reference Publication (60):
Agha et al., Actorspaces: An Open Distributed Programming Paradigm, University of Illinois, Report No. UIUCDCS-R-92-1766, Open Systems Laboratory TR No. 8, Nov. 1992, pp. 1-12.

Other Reference Publication (62):
Liskov et al., Distributed Object Management in Thor, International Workshop on Distributed Object Management, 1992, pp. 12.

Other Reference Publication (63):
Coulouris et al., Distributed Systems Concepts and Designs, Second Edition, Addison-Wesley, 1994.

Other Reference Publication (65):
Birrell et al., Distributed Garbage Collection for Network Objects, DEC SRC Research Report 116, Dec. 15, 1993.

Other Reference Publication (67):
Wollrath et al., A Distributed Object Model for the Java.TM. System, USENIX Association, Conference on Object-Oriented Technologies and Systems, Jun. 17-21, 1996.

Other Reference Publication (68):
Harris et al., Proposal for a General Java Proxy Class for Distributed Systems and Other Uses, Netscape Communications Corp., Jun. 25, 1997.

Other Reference Publication (82):
Dave, Amitabh, et al., "Proxies, Application Interfaces, and Distributed System," 1992.

Other Reference Publication (84):
Fleisch, B.D. & Hyde, R.L., "High performance distributed objects using distributed shared memory and remote method invocation," Jan. 6-9, 1998.

Other Reference Publication (85):
Gray, Cary, et al., "Leases: An Efficien Fault0Tolerant Mechanism for Distributed File Cache Consistency," 1989.

Other Reference Publication (86):
Guyennet, H., et al., "Distributed Shared Memory Layer for Cooperative Work Applications," 1997.

Other Reference Publication (89):
Newmarch, J., "Chapter 3: Discovering a Lookup Service Contents," 1999.

Other Reference Publication (95):
"Taxonomy for Distributed Network Caches," http://ringer.cs.utsa.edu/.about.sdykes/taxonomy.html.

Other Reference Publication (96):
Burns et al., "An Analytical Study of Opportunistic Lease Renewal", Distributed Computing Systems, 21.sup.st International Conference, pp. 146-153, Apr. 2000.

Other Reference Publication (103):
Betz, Mark; "Interoperable objects: laying the foundation for distributed object computing"; Dr. Dobb's Journal, vol. 19, No. 11, p. 18(13); (Oct. 1994).

Other Reference Publication (104):
Bevan, D.I., "An Efficient Reference Counting Solution To The Distributed Garbage Collection Problem", Parall Computing, NL, Elsevier Publishers, Amsterdam, vol. 9, No. 2, pp. 179-192.

Other Reference Publication (105):
Dave A et al: "Proxies, Application Interface, and Distributed Systems", Proceedings International Workshop on Object Orientation in Operating Systems, Sep. 24, 1992, pp. 212-220.

Other Reference Publication (112):
Guth, Rob: "JavaOne: Sun to Expand Java Distributed Computing Effect", "http://www/sunworld.com/swol-02-1998/swol-02-sunspots.html." XP-002109935, p. 1, 1998.

Other Reference Publication (113):
Hamilton et al., "Subcontract: a flexible base for distributed programming"; Proceedings of 14th Symposium of Operating System Principles; (Dec. 1993).

Other Reference Publication (118):
IBM: "Chapter 6--Distributed SOM (DSOM)" SOMObjects Developer Toolkit Users Guide, Version 2.1, (Oct. 1994), pp. 6-1-6-90.

Other Reference Publication (124):
Orfali R. et al., "The Essential Distributed Objects Survival Guide," Chapter 11: Corba Commercial ORBs, pp. 203-215, John Wiley & Sons, Inc., (1996).

Other Reference Publication (128):
Waldo J et al: "Events in an RPC based distributed system" Proceedings of the 1995 Usenix Technical Conference, Proceedings Usenix Winter 1995 Technical Conference, New Orleans, LA. USA, Jan. 16-20, 1995, pp. 131-142.

Other Reference Publication (131):
Yemini, Y. and S. da silva, "Towards Programmable Networks", IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, L'Aquila, Italy, 10/96.

CLAIMS:

1. A method for controlling program execution in a distributed computer system comprising the steps of: registering interest in an occurrence of an event in the distributed computer system, the registration of interest including information identifying the occurrence of the event, an identifier of a software entity in the distributed system, and a first object including a process and parameter data corresponding to the process; monitoring at least a portion of the distributed computer system for the occurrence of the registered event; and notifying the software entity identified in the registration of interest when the event occurs, the notification including a copy of the first object and an identification of the event that occurred.

3. The method of claim 1, wherein the step of registering interest further includes registering interest by a process located in a virtual machine different than the virtual machine in which the software entity is located.

4. The method of claim 1, wherein the step of monitoring the distributed computer system is performed by a process in a virtual machine different than the virtual machine in which the software entity is located.

5. The method of claim 1, wherein the step of registering interest further includes registering interest by a registering process located in an address space different from an address space in which the software entity is located.

6. The method of claim 1, wherein the step of registering interest further includes registering interest by a registering process and the step of monitoring includes a monitoring process, wherein the registering process is located in an address space different from an address space in which the monitoring process is located.

7. The method of claim 1, wherein the step of monitoring the distributed computer system is performed by a registering process in an address space different from an address space in which the software entity is located.

8. The method of claim 1, wherein the step of registering interest includes registering interest in an occurrence of a change in system state.

9. The method of claim 1, wherein the step of registering interest includes registering

interest in an occurrence of one selected from the group consisting of a timer event, a mouse click event, and a disk access event.

12. The method of claim 10, wherein the step of registering interest further includes registering interest by a process located in a virtual machine different from the virtual machine in which the software entity is located.

13. The method of claim 10, wherein the step of monitoring the distributed computer system is performed by a process in a virtual machine different from the virtual machine in which the software entity is located.

14. A protocol for controlling the execution of processes in a distributed computer system, the protocol comprising the steps of: receiving a registration of interest in an event that is expected to occur in the distributed computer system, the registration including an identifier of a software entity in the distributed system and a first object, the first object including computer instructions for performing a process and parameter data corresponding to the process; monitoring the distributed system for the occurrence of the registered event; and notifying the software entity identified in the registration of interest when the event occurs, the notification including a copy of the first object and an identification of the event that occurred.

17. The protocol of claim 14, wherein the step of monitoring the distributed system is performed by a process in an address space different from an address space in which the software entity is located.

18. The method of claim 14, wherein the step of registering interest further includes registering interest by a registering process and the step of monitoring includes a monitoring process, wherein the registering process is located in an address space different from an address space in which the monitoring process is located.

21. A computer readable medium containing instructions for controlling program execution in a distributed computer system, the instructions causing the distributed computer system to perform the steps of: registering interest in an event in the distributed computer system, the registration of interest including information identifying the event, an identifier of a software entity in the distributed system, and computer code for executing a process; monitoring at least a portion of the distributed computer system for the occurrence of the registered event; notifying the software entity identified in the registration of interest when the event occurs, the notification including a copy of the computer code and an identification of the event that occurred; and executing methods contained within the computer code in response to the notifying step.

22. A computer readable medium containing instructions for implementing a protocol for controlling the execution of processes in a distributed computer system, the instructions causing a computer in the distributed computer system to perform the steps of: receiving a registration of interest in an event in the distributed computer system, the registration including an identifier of a software entity in the distributed system and a first object, the first object including computer instructions for performing a process and parameter data corresponding to the process; monitoring the distributed system for the occurrence of the registered event; and notifying the software entity identified in the registration of interest when the event occurs, the notification including a copy of the first object and an identification of the event that occurred.

L13: Entry 14 of 90 File: USPT Aug 27, 2002

DOCUMENT-IDENTIFIER: US 6442564 B1
TITLE: Facilitating workload management by using a location forwarding capability

DATE FILED (1):
19990614

Brief Summary Text (2):
This invention relates, in general, to object-oriented computing environments and, in particular, to providing a distributed, object-oriented computing environment that is reliable, secure, transactional and workload managed.

Brief Summary Text (7):
One goal of the OMG is to provide distributed object-oriented applications and systems that coincide with the needs and desires of the ever-changing computing industry. This goal includes supporting multi-vendor, global heterogeneous networks.

Brief Summary Text (8):
Although efforts have been made to meet the goals of the Object Management Group, and of the object-oriented industry as a whole, further enhancements are still needed. For example, a need exists for a distributed object-oriented computing environment that is reliable, secure, transactional and workload managed.

Brief Summary Text (10):
The shortcomings of the prior art are overcome and additional advantages are provided through the provision of a method of facilitating workload management of a computing environment. The computing environment includes a plurality of server instances, and the method includes, for example, receiving, by a location service agent of a server instance of the plurality of server instances, a request for an object of the distributed computing environment; and having the location service agent request a workload manager of the server instance to determine which server instance of the plurality of server instances is to handle the request.

Brief Summary Text (14):
In a further aspect of the present invention, a system of facilitating workload management of a computing environment is provided. The computing environment includes a plurality of server instances, and the system includes, for instance, a location service agent of a server instance of the plurality of server instances adapted to receive a request for an object of the distributed computing environment; and the location service agent being adapted to request a workload manager of the server instance to determine which server instance of the plurality of server instances is to handle the request.

Brief Summary Text (16):
In another aspect of the present invention, an article of manufacture including at least one computer usable medium having computer readable program code means embodied therein for causing the facilitating of workload management of a computing environment is provided. The computer readable program code means in the article of manufacture includes, for instance, computer readable program code means for causing a computer to receive, by a location service agent of a server instance of the plurality of server instances, a request for an object of the distributed computing environment; and computer readable program code means for causing a computer to have the location service agent request a workload manager of the server instance to determine which server instance of the plurality of server instances is to handle the request.

Drawing Description Text (19):
FIG. 16a depicts one example of a distributed name space, in accordance with the principles of the present invention;

Drawing Description Text (20):
FIG. 16b depicts one example of a non-distributed name space, in accordance with the principles of the present invention;

Drawing Description Text (34):
FIG. 30 depicts one embodiment of the logic associated with registering multiple interfaces for a particular implementation, in accordance with the principles of the present invention; and

Detailed Description Text (2):
In accordance with the principles of the present invention, an infrastructure is provided that supports an object-oriented, component-based programming model and provides for a computing environment that is distributed, reliable, secure, transactional, and workload managed.

Detailed Description Text (41):
In accordance with one aspect of the present invention, a set of management policies, selectable by the customer at object installation time is provided (e.g., when the container is defined). These policies govern the management of state icoherency, isolation level and residence lifetimes of both transient and persistent objects in the virtual memory of a distributed object server, such as server instance 114 (see FIG. 7). The policies 700 are managed by one or more containers 118 and include, for instance, an activation isolation policy, a passivation policy, a flush policy and a refresh policy, each of which is described below.

Detailed Description Text (64):
In addition to the above, during object activation, the container also registers itself with an object transaction service (OTS) 900 (see FIG. 9) as an OTS synchronization object. (OMG OTS is described in CORBA services specification available on the OMG web page (WWW.OMG.ORG) under technical library, and is hereby incorporated herein by reference in its entirety.) That is, the container places various information in a table associated with OTS. As shown in FIG. 9, OTS is coupled to a resource recovery service (RRS) 902 located within the operating system. Since the container is a synchronization object, it implements a "before completion" method and "after completion" method. Thus, as one example, when a transaction is ready to be committed, OTS uses the sync object to inform its objects that a commit is about to happen ("before completion") and that it has happened ("after completion").

Detailed Description Text (98):
DB2 also knows that by attachment through RRSAF that DB2 has some special exits. DB2 registers its interest in this transaction using resource recovery service 1116, which is tracking the transaction. (RRSAF is a part of that service.)

Detailed Description Text (100):
Within a distributed object server instance 1200 (FIG. 12a), managed or business objects 1202 are often persistent. That is, they are composed from data residing in one or more databases 1204. In a legacy environment, where there exists a variety of data sources, such as DB2, IMS and CICS (all offered by International Business Machines Corporation, Armonk, N.Y.), as well as various other data sources, there is a need to extract data from these resource managers and use the data to provide a persistent object with its essential state.

Detailed Description Text (111):
In one embodiment, when a container sees a transaction for the first time, the container registers a synchronization object with OTS and also informs each connection object that the transaction is to be identified. Depending on the type of attachment, that connection object may or may not perform an action. For example, for DB2, the connection object creates a DB2 representative of a thread and signs the user on, so that DB2 knows about the user. In particular, in the connection object, the transaction is identified and DB2 creates a thread and anchors one of its structures onto the RRS context. Thus, when a data object accesses DB2, the data object pulls its own data out of the context.

Detailed Description Text (121):
Within the IIOP protocol, there exists an architected control flow that occurs when a client attempts to locate an object of interest in a distributed network. This control flow is defined by CORBA and is referred to as the "locate flow". There are three possible responses to this flow. The flow occurs to the server instance identified by

the object reference used to identify the object, and the server instance responds with: 1) the object is here on the target server instance, 2) the object does not exist, or 3) the object is not here on the server instance, but it may be on another server identified by the new object reference being returned with this response. This third type of response is called a "location forward" reply. The client ORB on receiving the location forward type of reply issues another locate request to the server instance identified by the returned object reference.

Detailed Description Text (122):
In accordance with one aspect of the present invention, the location forwarding mechanism is used in a unique way to cause the balancing of client communication endpoints across a set of replicated server instances. The mechanism uses, for instance, direct and indirect object references. A direct object reference is one in which the actual network address of the subject server process (address space) is bound. An indirect object reference is one in which the network address of a location service agent or daemon is bound. It is assumed for this aspect of the present invention that first class references to distributed objects are indirect. Thus, any reference obtained by a distributed client application is indirect and results in a locate request to the location service daemon when it is used.

Detailed Description Text (125):
In order for the location service agent to be able to have the workload manager select the appropriate server instance to perform a task, the agent first registers itself with the workload manager as a daemon. This has specific meaning to the workload manager. In particular, the workload manager expects the daemon to come back to the workload manager to request the workload manager to locate a particular environment name.

Detailed Description Text (126):
In order to register with the workload manager, the daemon uses various workload management interfaces to obtain the best location of a server instance that has registered with the workload manager. It is the server instance (e.g., a control region, as described below) that registers with WLM as a "work manager." The daemon asks WLM to find an appropriate "work manager" under a given class of name, and WLM returns the location.

Detailed Description Text (127):
The use of replicated server instances is optimal when all of the server instances have equal and shared access to the resources required by the application server instance. Two specific advantages realized when this replication is enabled are 1) reduction of single points of failure, 2) balanced performance and scalability. Further, the 390 system design asserts that the optimal place to make workload management decisions regarding the placement of inbound work over a set of replicated server instances is at the sysplex facility where all pertinent information required for making an intelligent decision of the placement of the workload exists and can be evaluated in real time as the work arrives to the configuration. In addition, in a distributed system, where communication protocols and the runtimes implementing those protocols are standardized to allow the communication to occur across a set of clients and servers provided by different vendors, there is an advantage in making workload management decisions at the server side of the communication runtime so that proprietary workload management based extensions are not required in the various client side runtimes provided by all of the various vendors.

Detailed Description Text (129):
Within a distributed system, and within a prescribed unit of application activity, there may be several interactions from a given client either directly to a target server instance where such application state is being held, or through multiple other intermediate middle tier server instances to the target server. All requests to the target server's application state under a given application unit of activity is directed to the same physical instance of the server's address space in order to ensure proper execution of the distributed application. If, however, the target server instance is replicated across the physical systems in a clustered configuration, and access to the replicated server instances is governed by workload management policy, sessions from intermediate nodes in the distributed network, established to the target server instance under a common unit of application activity may be balanced or spread to different physical server instances, thereby introducing errant behavior in the application. In other words, workload balancing across a set of replicated servers is to occur at well known boundaries within the execution of the application. Within those boundaries, a mechanism is used to ensure a temporary affinity to a specific server

instance within a clustered system configuration is defined, recognized and enforced, so that any requests under the same unit of application activity from any node in the distributed network arrives at the same physical server instance in the configuration.

Detailed Description Text (131):
One embodiment of ensuring a given unit of work arrives at an appropriate server instance is described with reference to FIG. 15. When a method request arrives at a server instance, and it is accompanied by a particular unit of work, such as a distributed transactional context, STEP 1500, a determination is made, by the ORB, as to whether the server instance receiving the request is the owner of the transaction, INQUIRY 1502. In other words, the server instance makes a local decision as to whether it already owns responsibility for the inbound transaction. In one example, this determination is made by checking a registration table located within the coupling facility coupled to the server instance or within memory of the server instance.

Detailed Description Text (133):
If the server instance determines that it is not the registered owner of the transaction, it attempts to register the server's interest in the inbound transaction, STEP 1504. This registration occurs, for example, at the coupling facility and is performed in an atomic fashion by using, for example, a Global Resource Serialization (GRS) global enqueue (ENQ). GRS is described in "OS/390 MVS Planning: Global Resource Serialization," IBM Pub. No. GC28-1759-04 (March 1998); "OS/390 MVS Programming: Authorized Assembler Services Guide," IBM Pub. No. GC28-1763-06 (March 1999); "OS/390 MVS Programming: Authorized Assembler Services Reference," IBM Pub. Nos. GC28-1764-05 (September 1998), GC28-1765-07 (March 1999), GC28-1766-05 (March 1999), and GC28-1767-06 (December 1998); "OS/390 MVS Programming: Assembler Services Guide," IBM Pub. No. GC28-1762-01 (September 1996) and "OS/390 MVS Programming: Assembler Services Reference," IBM Pub. No. GC28-1910-1 (September 1996), each of which is hereby incorporated herein by reference in its entirety. The registration information includes, for instance, specified user data (e.g., a UUID) in the request, which identifies the specific server instance within the configuration associated with the specified transaction.

Detailed Description Text (134):
Thereafter, a determination is made as to whether the registration was successful, INQUIRY 1506. That is, if no other server instance replica has registered its interest in the same transaction, then registration succeeds and that server is deemed the appropriate server instance for the unit of work, STEP 1508. If some other 'server instance replica has registered its interest in the transaction, the attempt to register fails. In this case, a GRS operation (e.g., a GQSCAN) is initiated to obtain the registration information, which includes the identity of the server instance that is registered for the transaction and the transaction id, STEP 1510. When that information is returned, a new object reference is built using the location of the registered server instance and the current object key for the object being dispatched, STEP 1512.

Detailed Description Text (135):
Subsequently, the server instance returns a location forwarding reply to the client, STEP 1514. The CORBA protocol allows the location forwarding reply to be returned on any distributed method request sent to a given server. The client ORB then uses the new object reference to establish an IIOP connection to the appropriate server instance within the sysplex, STEP 1516.

Detailed Description Text (140):
A fundamental concept in the OMG Naming Service is that the name space is composed of naming contexts bound together to form a tree 1604. Since each naming context is itself an object, and thus, may be distributed, the name space may be distributed across multiple name servers across the distributed network. (As another example, the name space is not distributed (see FIG. 16b)).

Detailed Description Text (149):
Defining the naming context in this manner allows efficient one hop lookup of objects that reside in a particular name server. Since each object that is included in the name space is represented by a binding object, the retrieval of a named binding of a given name requires a search against a single home collection--the binding home collection.

Detailed Description Text (158):
As described above, in constructing the name space, it is possible that portions of the name space may exist in different physical systems. That is, one server may be on one

system, while another server may be on another system. For example, with reference to FIG. 17, Naming Context `/B` may exist on System A, while Naming Context `/B/D` may exist on System B. The underlying resource managers for these naming contexts may not be the same (e.g., they may have different data schemas and formats in the database, different protocols used to navigate the directory schema, and/or different formats of the names used to represent the entries in the directory system, etc.). For example, naming context `/B` may be stored in LDAP, and Naming Context `/B/D` may be stored in DCE CDS (Distributed Computer Environment Cell Directory Services). A schematic illustration of this scenario is depicted in FIG. 19, in which a Naming Context `/B` 1900 is located in Name Server 1 (1902) of System A and is backed by an LDAP resource manager 1904; and Naming Context `/B/D` 1906 is located in Name Server 1 (1908) of System B and its data is backed by a DCE CDS resource manager 1910.

Detailed Description Text (223):
One embodiment of the logic associated with registering multiple interfaces for the factory of implementation E is described with reference to FIG. 30. Initially, a transaction is begun for performing the registrations, STEP 3000. Within the transactional unit of work, the factory for implementation E, in this example, is registered under interface Name A, STEP 3002. In particular, the factory object is bound in the name space under Name A.

Detailed Description Text (236):
Another flaw with single address space server instances is poor transaction recovery time. If the distributed application server provides the capability of managing work under a transactional unit of recovery, then the server is also obligated to provide well known recovery actions in the case of failure. One of the pieces of recovery after server failure is the re-establishment of the distributed communication channels that were in operation at the time of failure. This recovery action is used to determine whether or not any further recovery action is necessary or desired. In addition, the transaction log which provides information regarding the state of the transaction in the server at time of failure is read. This log is typically tied to the server address space that failed. This means that each and every address space is to be restarted, replay its transaction log, and then determine whether further action is required. As the number of server address spaces increase in the system, restart time becomes an inhibitor to the efficient and responsive recovery of the system.

CLAIMS:

1. A method of facilitating workload management of a computing environment, said computing environment comprising a plurality of server instances, and said method comprising: receiving, by a location service agent of a server instance of said plurality of server instances, a request for an object of said distributed computing environment; and having said location service agent request a workload manager of said server instance to determine which server instance of said plurality of server instances is to handle said request.

11. A system of facilitating workload management of a computing environment, said computing environment comprising a plurality of server instances, and said system comprising: a location service agent of a server instance of said plurality of server instances adapted to receive a request for an object of said distributed computing environment; and said location service agent being adapted to request a workload manager of said server instance to determine which server instance of said plurality of server instances is to handle said request.

21. A system of facilitating workload management of a computing environment, said computing environment comprising a plurality of server instances, and said system comprising: means for receiving, by a location service agent of a server instance of said plurality of server instances, a request for an object of said distributed computing environment; and means for having said location service agent request a workload manager of said server instance to determine which server instance of said plurality of server instances is to handle said request.

23. An article of manufacture, comprising: at least one computer usable medium having computer readable program code means embodied therein for causing the facilitating of workload management of a computing environment, said computing environment comprising a plurality of server instances, and the computer readable program code means in said article of manufacture comprising: computer readable program code means for causing a computer to receive, by a location service agent of a server instance of said plurality of server instances, a request for an object of said distributed computing environment;

and computer readable program code means for causing a computer to have said location service agent request a workload manager of said server instance to determine which server instance of said plurality of server instances is to handle said request.

WEST

Generate Collection

L13: Entry 18 of 90 File: USPT May 21, 2002

DOCUMENT-IDENTIFIER: US 6393497 B1
TITLE: Downloadable smart proxies for performing processing associated with a remote procedure call in a distributed system

DATE FILED (1):
19980320

Abstract Text (1):
Use of a smart proxy as a wrapper around a stub in a distributed system. Instead of receiving a stub as a result of a remote procedure call, a caller receives a smart proxy including the stub as an embedded object. The smart proxy performs predefined processing associated with a remote procedure call, the processing possibly occurring before, during, or after a response to the call.

Parent Case Text (2):
Provisional U.S. patent application No. 60/076,048, entitled "Distributed Computing System," filed on Feb. 26, 1998.

Parent Case Text (4):
U.S. patent application Ser. No. 09/044,838, entitled "Method, Apparatus, and Product for Leasing of Delegation Certificates in a Distributed System," filed on the same date herewith.

Parent Case Text (5):
U.S. patent application Ser. No. 09/044,834, entitled "Method, Apparatus and Product for Leasing of Group Membership in a Distributed System," filed on the same date herewith.

Parent Case Text (8):
U.S. patent application Ser. No. 09/044,919, entitled "Deferred Reconstruction of Objects and Remote Loading for Event Notification in a Distributed System," filed on the same date herewith.

Parent Case Text (11):
U.S. patent application Ser. No. 09/044,790, entitled "Method and Apparatus for Determining Status of Remote Objects in a Distributed System," filed on the same date herewith.

Parent Case Text (16):
U.S. patent application Ser. No. 09/044,931, entitled "Dynamic Lookup Service in a Distributed System," filed on the same date herewith.

Parent Case Text (17):
U.S. patent application Ser. No. 09/044,939, entitled "Apparatus and Method for Providing Downloadable Code for Use in Communicating with a Device in a Distributed System," filed on the same date herewith.

Parent Case Text (18):
U.S. patent application Ser. No. 09/044,826, entitled "Method and System for Facilitating Access to a Lookup Service," filed on the same date herewith.

Parent Case Text (19):
U.S. patent application No. 09/030,840, entitled "Method and Apparatus for Dynamic Distributed Computing Over a Network, " and filed on Feb. 26, 1998.

Parent Case Text (20):
U.S. patent application Ser. No. 09/044,932, entitled "Apparatus and Method for

Dynamically Verifying Information in a Distributed System," filed on the same date herewith.

Brief Summary Text (2):
The present invention relates to a system and method for transmitting objects between machines in a distributed system and more particularly relates to transmission of a representation of a remote object including code for local processing.

Brief Summary Text (4):
Distributed programs which concentrate on point-to-point data transmission can often be adequately and efficiently handled using special-purpose protocols for remote terminal access and file transfer. Such protocols are tailored specifically to the one program and do not provide a foundation on which to build a variety of distributed programs (e.g., distributed operating systems, electronic mail systems, computer conferencing systems, etc.).

Brief Summary Text (5):
While conventional transport services can be used as the basis for building distributed programs, these services exhibit many organizational problems, such as the use of different data types in different machines, lack of facilities for synchronization, and no provision for a simple programming paradigm.

Brief Summary Text (6):
Distributed systems usually contain a number of different types of machines interconnected by communications networks. Each machine has its own internal data types, its own address alignment rules, and its own operating system. This heterogeneity causes problems when building distributed systems. As a result, program developers must include in programs developed for such heterogeneous distributed systems the capability of ensuring that information is handled and interpreted consistently in different machines.

Brief Summary Text (16):
Because the JVM may be implemented on any type of platform, implementing distributed programs using the JVM significantly reduces the difficulties associated with developing programs for heterogenous distributed systems. Moreover, the JVM uses a Java remote method invocation system (RMI) that enables communication among programs of the system. RMI is explained in, for example, the following document, which is incorporated herein by reference: Remote Method Invocation Specification, Sun Microsystems, Inc. (1997), which is available via universal resource locator (URL)

Brief Summary Text (18):
FIG. 2 is a diagram illustrating the flow of objects in an object-oriented distributed system 200 including machines 201 and 202 for transmitting and receiving method invocations using the JVM. In system 200, machine 201 uses RMI 205 for responding to a call for object 203 by converting the object into a byte stream 207,including an identification of the type of object transmitted and data constituting the object. While machine 201 is responding to the call for object 203, a process running on the same or another machine in system 200 may continue operation without waiting for a response to its request.

Drawing Description Text (4):
FIG. 2 is a diagram illustrating the transmission of objects in an object-oriented distributed system;

Drawing Description Text (5):
FIG. 3 is a diagram of an exemplary distributed processing system that can be used in an implementation consistent with the present invention;

Drawing Description Text (6):
FIG. 4 is a diagram of an exemplary distributed system infrastructure;

Drawing Description Text (7):
FIG. 5 is a diagram of a computer in a distributed system infrastructure shown in FIG. 4;

Drawing Description Text (8):
FIG. 6 is a block diagram of a distributed network for use in downloading smart proxies;

Drawing Description Text (9):
FIG. 7 is a flow chart of a process for downloading smart proxies within, for example, the distributed network shown in FIG. 6; and

Detailed Description Text (3):
Instead of receiving a proxy that only makes network requests to the object for which it is a surrogate, a machine in a distributed system receives a smart proxy. Such a proxy can respond to calls on the object for which it is a surrogate without making any network calls to increase program efficiency, or perform processing before making a network call or after the completion of the network call to increase program functionality. The term proxy generally refers to code or other mechanism used to act as a surrogate for a remote object in the address space of a machine.

Detailed Description Text (4):
Systems transferring stubs and associated smart proxies may/use a variant of an RPC or RMI, passing arguments and return values from one process to another process each of which may be on different machines. The term "machine" is used in this context to refer to a physical machine or a virtual machine. Multiple virtual machines may exist on the same physical machine. Examples of RPC systems include distributed computed environment (DCE) RPC and Microsoft distributed common object model (DCOM) RPC. A memory stores the stub and associated smart proxy, and this memory may include secondary sources such as a disk or receiving objects from the Internet.

Detailed Description Text (5):
Distributed Processing System

Detailed Description Text (6):
FIG. 3 illustrates an exemplary distributed processing system 300 which can be used in an implementation consistent with the present invention. In FIG. 3, distributed processing system 300 contains three independent and heterogeneous platforms 301, 302, and 303 connected in a network configuration represented by network cloud 319. The composition and protocol of the network configuration represented by cloud 319 is not important as long as it allows for communication of the information between platforms 301, 302 and 303. In addition, the use of just three platforms is merely for illustration and does not limit an implementation consistent with the present invention to the use of a particular number of platforms. Further, the specific network architecture is not crucial to embodiments consistent with this invention. For example, another network architecture that could be used in an implementation consistent with this invention would employ one platform as a network controller to which all the other platforms would be connected.

Detailed Description Text (7):
In the implementation of distributed processing system 300, platforms 301, 302 and 303 each include a processor 316, 317, and 318 respectively, and a memory, 304, 305, and 306, respectively. Included within each memory 304, 305, and 306, are applications 307, 308, and 309, respectively, operating systems 310, 311, and 312, respectively, and RMI components 313, 314, and 315, respectively.

Detailed Description Text (12):
Distributed System Infrastructure

Detailed Description Text (13):
Systems and methods consistent with the present invention may also operate within a particular distributed system 400, which will be described with reference to FIGS. 4 and 5. This distributed system 400 is comprised of various components, including hardware and software, to (1) allow users of the system to share services and resources over a network of many devices; (2) provide programmers with tools and programming patterns that allow development of robust, secured distributed systems; and (3) simplify the task of administering the distributed system. To accomplish these goals, distributed system 400 utilizes the Java programming environment to allow both code and data to be moved from device to device in a seamless manner. Accordingly, distributed system 400 is layered on top of the Java programming environment and exploits the characteristics of this environment, including the security offered by it and the strong typing provided by it.

Detailed Description Text (14):
In distributed system 400 of FIGS. 4 and 5, different computers and devices are federated into what appears to the user to be a single system. By appearing as a single system, distributed system 400 provides the simplicity of access and the power of

sharing that can be provided by a single system without giving up the flexibility and personalized response of a personal computer or workstation. Distributed system 400 may contain thousands of devices operated by users who are geographically disperse, but who agree on basic notions of trust, administration, and policy.

Detailed Description Text (15):
Within an exemplary distributed system are various logical groupings of services provided by one or more devices, and each such logical grouping is known as a Djinn. A "service" refers to a resource, data, or functionality that can be accessed by a user, program, device, or another service and that can be computational, storage related, communication related, or related to providing access to another user. Examples of services provided as part of a Djinn include devices, such as printers, displays, and disks; software, such as programs or utilities; information, such as databases and files; and users of the system.

Detailed Description Text (17):
Distributed system 400 is comprised of computer 402, a computer 404, and a device 406 interconnected by a network 408. Device 406 may be any of a number of devices, such as a printer, fax machine, storage device, computer, or other devices. Network 408 may be a local area network, wide area network, or the Internet. Although only two computers and one device are depicted as comprising distributed system 400, one skilled in the art will appreciate that distributed system 400 may include additional computers or devices.

Detailed Description Text (18):
FIG. 5 depicts computer 402 in greater detail to show a number of the software components of distributed system 400. One skilled in the art will appreciate that computer 404 or device 406 may be similarly configured. Computer 402 includes a memory 502, a secondary storage device 504, a central processing unit (CPU) 506, an input device 508, and a video display 510. Memory 502 includes a lookup service 512, a discovery server 514, and a Java runtime system 516. The Java runtime system 516 includes the Java RMI system 518 and a JVM 520. Secondary storage device 504 includes a Java space 522.

Detailed Description Text (19):
As mentioned above, distributed system 400 is based on the Java programming environment and thus makes use of the Java runtime system 516. The Java runtime system 516 includes the Java API libraries, allowing programs running on top of the Java runtime system to access, in a platform-independent manner, various system functions, including windowing capabilities and networking capabilities of the host operating system. Since the Java API libraries provide a single common API across all operating systems to which the Java runtime system is ported, the programs running on top of a Java runtime system run in a platform-independent manner, regardless of the operating system or hardware configuration of the host platform. The Java runtime system 516 is provided as part of the Java software development kit available from Sun Microsystems, Inc. of Mountain View, calif.

Detailed Description Text (21):
Lookup service 512 defines the services that are available for a particular Djinn. That is, there may be more than one Djinn and, consequently, more than one lookup service within distributed system 400. Lookup service 512 contains one object for each service within the Djinn, and each object contains various methods that facilitate access to the corresponding service. Lookup service 512 is described in U.S. patent application entitled "Method and System for Facilitating Access to a Lookup Service," which was previously incorporated herein by reference.

Detailed Description Text (22):
Discovery server 514 detects when a new device is added to distributed system 400, during a process known as boot and join (or discovery), and when such a new device is detected, the discovery server passes a reference to lookup service 512 to the new device so that the new device may register its services with the lookup service and become a member of the Djinn. After registration, the new device becomes a member of the Djinn, and as a result, it may access all the services contained in lookup service 512. The process of boot and join is described in U.S. patent application entitled "Apparatus and Method for providing Downloadable Code for Use in Communicating with a Device in a Distributed System," which was previously incorporated herein by reference.

Detailed Description Text (23):

A Java space 522 is an object repository used by programs within distributed system 400 to store objects. Programs use a Java space 522 to store objects persistently as well as to make them accessible to other devices within distributed system 400. Java spaces are described in U.S. patent application Ser. No. 08/971,529, entitled "Database System Employing Polymorphic Entry and Entry Matching," assigned to a common assignee, and filed on Nov. 17, 1997, which is incorporated herein by reference. One skilled in the art will appreciate that an exemplary distributed system 400 may contain many lookup services, discovery servers, and Java spaces.

Detailed Description Text (24):
Data Flow in a Distributed Processing System

Detailed Description Text (25):
FIG. 6 is a block diagram of an object-oriented distributed network 600 connecting machines 601 and 606, such as computers or virtual machines executing on one or more computers, or the machines described with reference to FIGS. 3, 4, and 5. Network 600 transmits proxies, some of which may be smart proxies. A smart proxy includes code for performing processing associated with a call. For example, a smart proxy may perform a caching operation for read-only data for later reference. When a call is made for that data, the smart proxy may obtain it locally and provide it to a user without making another call for the data, which may occur transparent to the user. An example of such read-only data is a particular installation time. The first time a call is made for the installation time, for example, a smart proxy locally caches that value, and when a subsequent call is made for the installation time, the smart proxy locally retrieves the value.

Detailed Description Text (27):
Other examples of uses of smart proxies include processing for debugging, call logging, and monitoring system performance. Another example involves the use of a smart proxy for local data verification, as explained in the U.S. patent application filed on the same date herewith, assigned to a common assignee, and entitled "Apparatus and Method for Dynamically Verifing Information in a Distributed System," which is incorporated herein by reference. Many other uses for smart proxies are possible for performing processing associated with a call.

Detailed Description Text (32):
When creating stub object 605, RMI 602 does not necessarily know that the stub is itself a smart proxy 604. Smart proxy 604 may perform processing at client machine 601 before or after response 610 and may supply all processing without resorting to call 609 to the object for which the proxy acts. Therefore, smart proxy 604 may perform all processing locally when client machine 601 makes a call or request 611 to invoke a method on smart proxy 604. These proxies are downloadable by the same methods as disclosed in U.S. patent application Ser. No. 08/950,756, filed on Oct. 15, 1997, and entitled "Deferred Reconstruction of Objects and Remote Loading in a Distributed System," which is incorporated herein by reference.

Detailed Description Text (34):
FIG. 7 is a flow chart of a process 700 for downloading and using smart proxies within, for example, the distributed network shown in FIG. 6. A client machine transmits a call or request for a particular object (step 701), and a server machine receives the call (step 702). In response, the server machine returns a smart proxy with an embedded stub (step 703), and the proxy acts as a representation of the requested object. After receiving the smart proxy, the client machine invokes a method on it (step 704). According to the code within the smart proxy, the client machine containing the smart proxy determines if preprocessing is required (step 705). If so, the processing is performed locally by the client machine using the smart proxy (step 706).

Other Reference Publication (3):
Aldrich et al., "Providing Easier Access to Remote Objects in Distributed Systems," Calif. Institute of Technology, www.cs.caltech.edu/%7Ejedi/paper/jedipaper.html, Nov. 21, 1997.

Other Reference Publication (4):
Burns et al., "An Analytical Study of Opportunistic Lease Renewal," Distributed Computing Systems, 21st International Conference, pp. 146-153, Apr. 2000.

Other Reference Publication (6):
Fleisch et al., "High Performance Distributed Objects Using Distributed Shared Memory & Remote Method Invocation," System Sciences, 1998, Proceedings of the 31st Hawaii

Internat'l. Conference, Jan. 6-9, 1998, pp. 574-578.

Other Reference Publication (7):
Gray et al., "Leases: An Efficient Fault-Tolerant Mechanism for Distributed File Cache Consistency," Proceedings of the 12th ACM Symposium on Operating Systems Principles, pp. 202-210, 1989.

Other Reference Publication (9):
Guyennet et al., "Distributed Shared Memory Layer for Cooperative Work Applications," IEEE, 0742-1303/97, pp. 72-78, 1997.

Other Reference Publication (17):
Mullender, Distributed Systems, Second Edition, Addison-Wesley, 1993.

Other Reference Publication (18):
Howard et al., Scale and Performance in a Distributed File System, ACM Transactions on Computer Systems, vol. 6, No. 1, Feb. 1988, pp. 51-81.

Other Reference Publication (20):
Dijkstra, Self-stabilizing Systems in Spite of Distributed Control, Communications of the ACM, vol. 17, No. 11, Nov. 1974, pp. 643-644.

Other Reference Publication (23):
Sharrott et al., ObjectMap: Integrating High Performance Resources into a Distributed Object-oriented Environment, ICODP, 1995.

Other Reference Publication (24):
Birrell et al., Grapevine: An Exercise in Distributed Computing, Communications of the ACM, vol. 25, No. 4, Apr. 1982, pp. 260-274.

Other Reference Publication (26):
Gray et al., Leases: An Efficient Fault-Tolerant Mechanism for Distributed File Cache Consistency, ACM, 1989, pp. 202-210.

Other Reference Publication (28):
Dolev et al., On the Minimal Synchronism Needed for Distributed Consensus, Journal of the ACM, vol. 34, No. 1, Jan. 1987, pp. 77-97.

Other Reference Publication (29):
Mummert et al., Long Term Distributed File Reference Tracing: Implementation and Experience, Carnegie Mellon University School of Computer Science, Nov. 1994, pp. 1-28.

Other Reference Publication (35):
Carriero et al., Distributed Data Structures in Linda, Principals of Programming Language, 1986, pp. 1-16.

Other Reference Publication (38):
Carriero et al, Distributed Data Structures in Linda, Yale Research Report YALEU/DCS/RR-438, Nov. 1985.

Other Reference Publication (39):
Agha et al., Actorspaces: An Open Distributed Programming Paradigm, University of Illinois, Report No. UIUCDCS-R-92-1766, Open Systems Laboratory TR No. 8, Nov. 1992, pp. 1-12.

Other Reference Publication (41):
Liskov et al., Distributed Object Management in Thor, International Workshop on Distributed Object Management, 1992, pp. 12.

Other Reference Publication (42):
Coulouris et al., Distributed Systems Concepts and Designs, Second Edition, Addison-Wesley, 1994.

Other Reference Publication (44):
Birrell et al., Distributed Garbage Collection for Network Objects, DEC SRC Research Report 116, Dec. 15, 1993.

Other Reference Publication (46):

Wollrath et al., A Distributed Object Model for the Java.TM. System, USENIX Association, Conference on Object-Oriented Technologies and Systems, Jun. 17-21, 1996.

Other Reference Publication (47):
Harris et al., Proposal for a General Java Proxy Class for Distributed Systems and Other Uses, Netscape Communications Corp., Jun. 25, 1997.

Other Reference Publication (64):
Betz, Mark; "Interoperable objects: laying the foundation for distributed object computing"; Dr. Dobb's Journal, vol. 19, No. 11, p. 18(13); (Oct. 1994).

Other Reference Publication (65):
Bevan, D.I., "An Efficient Reference Counting Solution To The Distributed Garbage Collection Problem", Parall Computing, NL, Elsevier Publishers, Amsterdam, vol. 9, No. 2, pp. 179-192, Jan. 1989.

Other Reference Publication (67):
Dave A et al: "Proxies, Application Interface, and Distributed Systems", Proceedings International Workshop on Object Orientation in Operating Systems, Sep. 24, 1992, pp. 212-220.

Other Reference Publication (74):
Guth, Rob: "JavaOne: Sun to Expand Java Distributed Computing Effort", "HTTP://WWW.SUNWORLD.COM/SWOL-02-1998/SWOL-02-SUNSPOTS.HTML," XP-002109935, p. 1, Feb. 20, 1998.

Other Reference Publication (75):
Hamilton et al., "Subcontract: a flexible base for distributed programming"; Proceedings of 14th Symposium of Operating System Principles; (Dec. 1993).

Other Reference Publication (87):
Waldo J et al: "Events in an RPC based distributed system" Proceedings of the 1995 USENIX Technical Conference, Proceedings USENIX Winter 1995 Technical Conference, New Orleans, LA. USA, Jan. 16-20, 1995, pp. 131-142.

Other Reference Publication (90):
Yemini, Y. and S. da silva, "Towards Programmable Networks" IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, L'Aquila, Italy, 10/96.

Other Reference Publication (92):
IBM: Somobjects Developer Toolkit Users Guide, Version 2.1, "Chapter 6 Distributed SOM (DSOM)," pp 6-1-6-90, Oct. 1994.

Other Reference Publication (93):
Orfali R. et al., "The Essential Distributed Objects Survival Guide," Chapter 11: Corba Commercial ORBs, John Wiley & Sons, Inc., (1996).

CLAIMS:

1. A method for receiving objects in a distributed system comprised of multiple machines, comprising:

transmitting a request for a particular object; and

receiving a response to the request, the response including code used to construct a representation of the requested object, the construction creating an object for processing calls to the particular object, local to the requesting object, using the representation.

4. A method for transmitting objects in a distributed system comprised of multiple machines, comprising:

receiving at a machine a request for a particular object; and

transmitting a response to the request, the response including first code for constructing a representation of the object and including an indication of second code for processing such that the construction creates an object for processing calls to the particular object, local to the requesting object, using the representation.

5. An apparatus for receiving objects in a distributed system comprised of multiple machines, comprising:

a module configured to transmit a request for a particular object; and

a module configured to receive a response to the request, the response including code used to construct a representation of the requested object, the construction creating an object for processing calls to the particular object, local to the requesting object, using the representation.

8. An apparatus for transmitting objects in a distributed system comprised of multiple machines, comprising:

a module configured to receive at a machine a request for a particular object; and

a module configured to transmit a response to the request, the response including first code for constructing a representation of the object and including an indication of second code for processing such that the construction creates an object for processing calls to the particular object, local to the requesting object, using the representation.

9. A system for transmitting objects in a distributed system comprised of multiple machines, comprising:

a first machine;

a second machine;

a network connecting the first machine with the second machine; and

an apparatus for receiving objects, the apparatus including:

a module configured to transmit a request for a particular object; and

a module configured to receive a response to the request, the response including code used to construct a representation of the requested object, the construction creating an object for processing calls to the particular object, local to the requesting object, using the representation.

12. A system for transmitting objects in a distributed system comprised of multiple machines, comprising:

a first machine;

a second machine;

a network connecting the first machine with the second machine; and an apparatus for transmitting objects, the apparatus including:

a module configured to receive at a machine a request for a particular object; and

a module configured to transmit a response to the request, the response including first code for constructing a representation of the object and including an indication of second code for processing such that the construction creates an object for processing calls to the particular object, local to the requesting object, using the representation.

17. An article of manufacture specifying a representation of an object stored in a computer-readable storage medium and capable of electronic transmission between machines in a distributed system, the article of manufacture comprising:

an object to be transmitted from a first machine to a second machine in a response to a request, the object including an indication of code used to construct a representation of the requested object, the construction creating an object for processing calls to the particular object, local to the requesting object, using the representation.

18. An apparatus for receiving objects in a distributed system comprised of multiple machines, comprising:

means for transmitting a request for a particular object; and

means for receiving a response to the request, the response including code used to construct a representation of the requested object, the construction creating an object for processing calls to the particular object, local to the requesting object, using the representation.